TÉCNICO
LISBOA

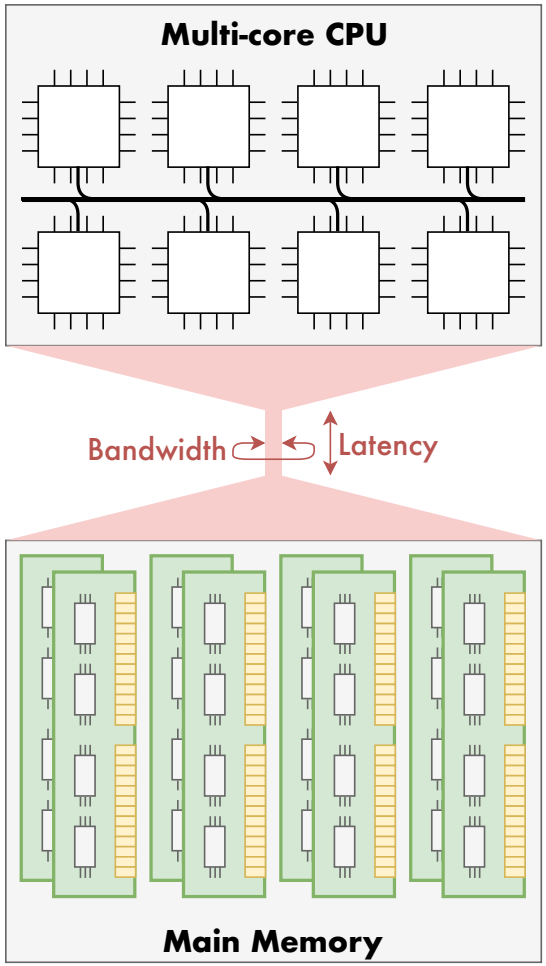
EXPLOITING PROCESSING NEAR CACHE FOR MEMORY BOUND VECTOR OPERATIONS

João Miguel Morgado Pereira Vieira

Thesis to obtain the Master of Science Degree in Electrical and Computer Engineering

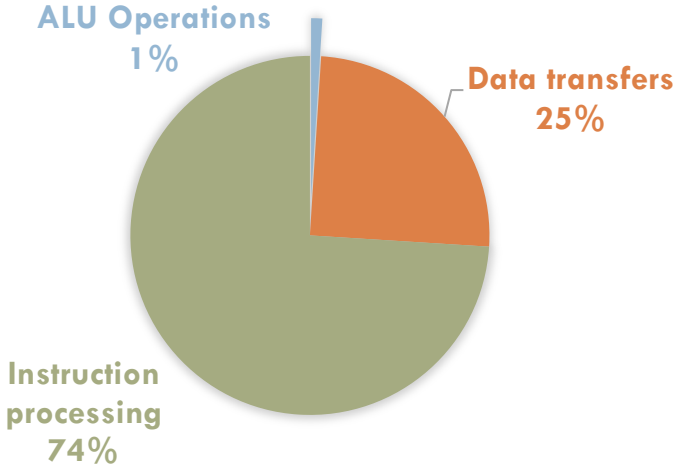
Supervisors:
Professor Doutor Pedro Filipe Zeferino Aidos Tomás
Professor Doutor Gabriel Falcão Paiva Fernandes

Motivation



- **The memory subsystem can limit throughput and energy performance:**
  - Bandwidth;
  - Latency.

Energy distribution of a processing core[1]



1. **R. Das** et al., "Compute caches," in *HPCA*. IEEE Computer Society, 2017, pp. 481–492;

Memory-intensive applications

- **Memory-intensive applications → negligible processing on big data sets**
  - Most clustering algorithms (e.g., kNN, K-Means, Mean Shift)
    - Euclidean distance: $d_{A,B} = \sqrt{\sum_i (a_i - b_i)^2}$
  - Parts of cryptographic algorithms (e.g., md5)
  - Basic algebra (e.g., vector addition)

```
for i from 0 to 63
    if 0 ≤ i ≤ 15 then
        f := (b and c) or ((not b) and d)
        g := i
    else if 16 ≤ i ≤ 31
        f := (d and b) or ((not d) and c)
        g := (5×i + 1) mod 16
    else if 32 ≤ i ≤ 47
        f := b xor c xor d
        g := (3×i + 5) mod 16
    else if 48 ≤ i ≤ 63
        f := c xor (b or (not d))
        g := (7×i) mod 16
```

It would be more efficient not to transfer the data to the core, but instead perform the computation in-place

State-of-the-art

- **Near-Data-Processing (NDP):**
  - **Processing-In-Memory (PIM)** solutions appeared during the 1980s[2]:
    - Mostly based on **bit-line processing;**
    - Hard to integrate before **3D-stacking** (introduced much later);
    - **Hard to use** (different programming paradigm).
  - Some solutions use custom hardware to perform computation near the memory[3]:
    - Not as efficient as bit-line computing;
    - Allow more **complex operations.**
  - These techniques were recently ported to the **cache**[1,4].

1. **R. Das** et al., "Compute caches," in *HPCA*. IEEE Computer Society, 2017, pp. 481–492;
2. **T. Mowry** et al., "Fast bulk bitwise AND and OR in DRAM", Computer Architecture Letters, vol. 14, no. 2, pp. 127–131, 2015;
3. **O. Mutlu** et al., "Pim-enabled instructions: a low-overhead, locality-aware processing-in-memory architecture," in ISCA. ACM, 2015, pp. 336–348;
4. **R. Das** et al., "Cache automaton," in *MICRO*. ACM, 2017, pp. 259–272.

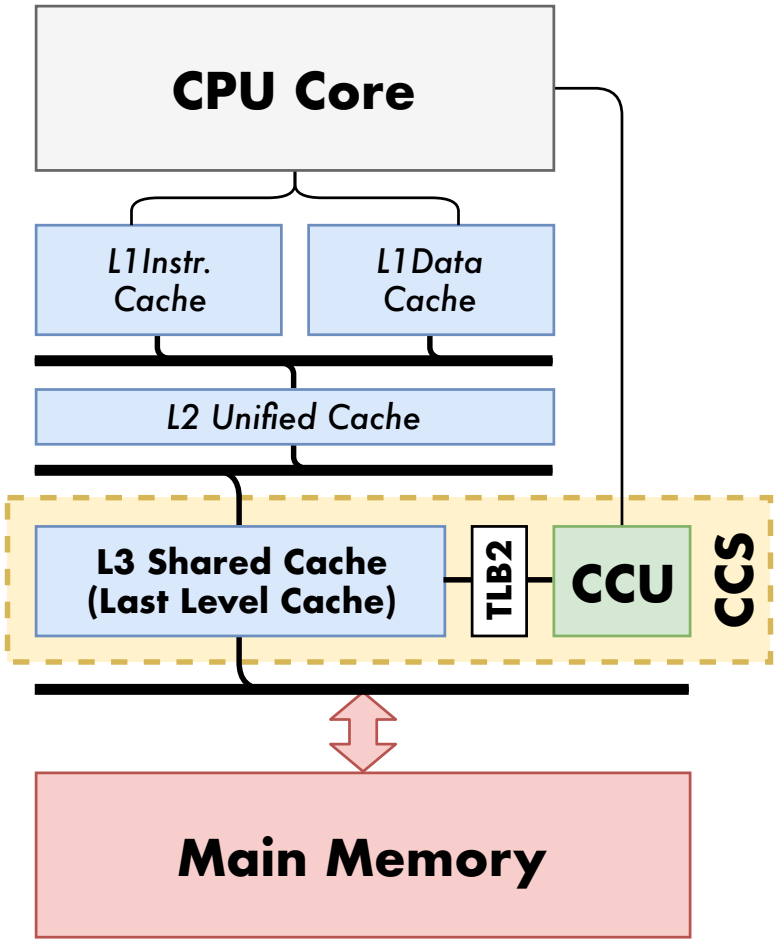
Objectives

- Stream-based architecture for a **cache Compute Unit;**
- Programming **Framework;**
- **Comparison with several processing systems;**
- **Evaluation** of the improvements:
  - **Performance;**
  - **Energy efficiency.**

Devised architecture

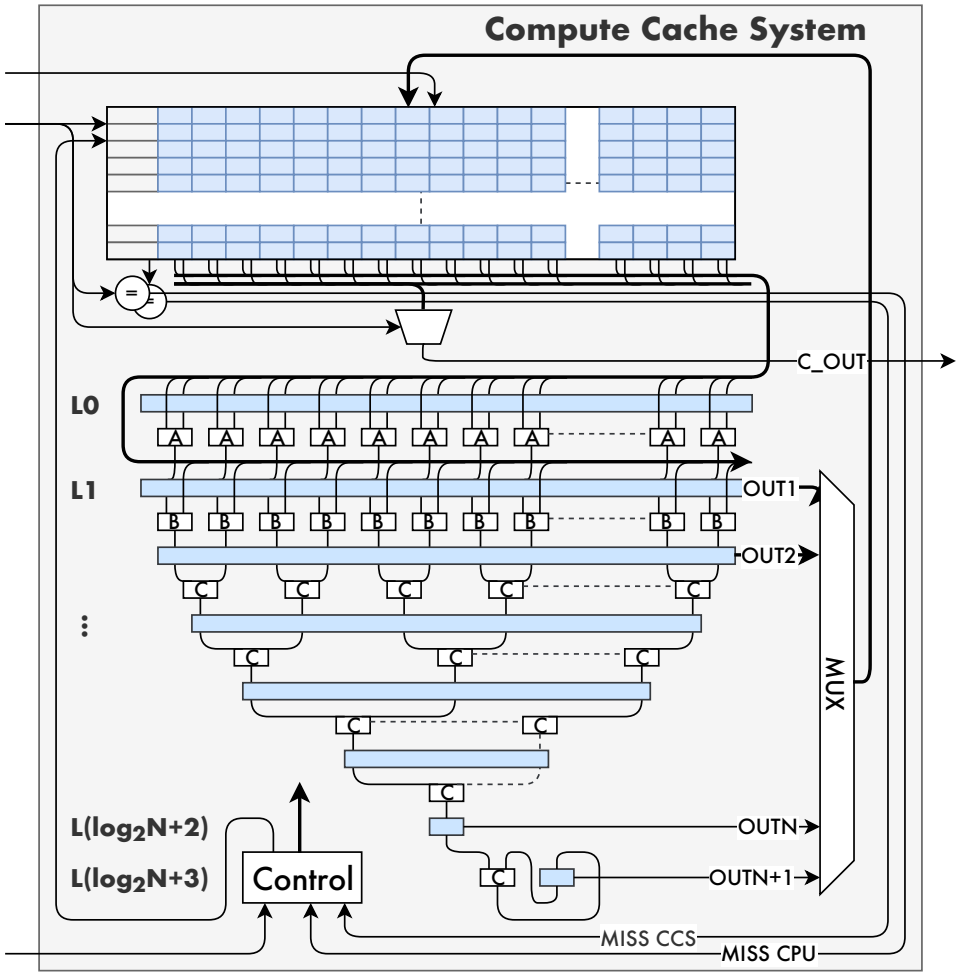
- Our **Compute Cache System (CCS)**:
  - Dedicated hardware to perform parallel computation near the cache, taking advantage of:
    - Operands locality;
    - Long cache lines;
    - Lower latency to the main memory;
    - Existing cache coherency protocols.
  - Optimized for:
    - **Map operations**: perform the same given operation in every element of a set (e.g., $c[i] = a[i] + b[i], i \in [0,63]$);
    - **Reduce operations**: apply an associative function to all elements of a set producing only 1 result (e.g., $c = \sum_i (a[i] - b[i])^2$).

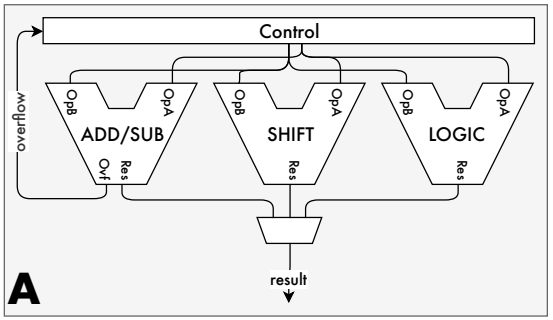
System schematics

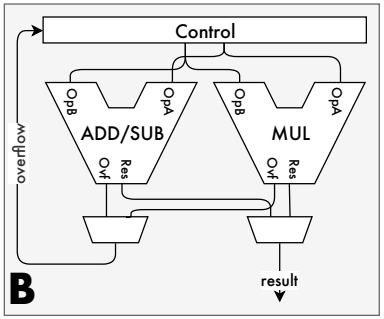


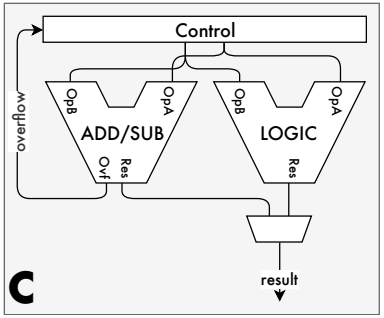
- **The CCS is integrated with an existing processing system:**
  - As a slave of a processing core;
  - Communicating with the LLC to get the operands;
  - Communicating with the main memory in case of cache miss.

Detailed architecture









Instruction Set Architecture (ISA)

Arithmetic

ADDVV	SUBVV	MULVV	ADDVC	SUBVC	MULVC	COMP2	ADDV	MAXV	MINV	LESSVC	GRTRVC	EQUVC	SQV	ABSV

Shift

SLLVV	SRLVV	SLAVV	SRAVV	ROLVV	RORVV	SLLVC	SRLVC	SLAVC	SRAVC	ROLVC	RORVC

Logic

ANDVV	NANDVV	ORVV	NORVV	XORVV	XNORVV	ANDVC	NANDVC	ORVC	NORVC	XORVC	XNORVC	NOTV	ANDV	ORV	XORV

Composed

SSDVV	SADVV	IPVV

Move

INITC	COPYV

Instruction Set Architecture (ISA)

Arithmetic

ADDVV	SUBVV	MULVV	ADDVC	SUBVC	MULVC					LESSVC	GRTRVC	EQUVC	SQV	ABSV

Shift

SLLVV	SRLVV	SLAVV	SRAVV	ROLVV	RORVV					ROLVC	RORVC			

Logic

ANDVV	NANDVV	ORVV	NORVV	XORVV	XNORVV	ANDVC	NANDVC	ORVC	NORVC	XORVC	XNORVC	NOTV	ANDV	ORV	XORV

Composed

SSDVV	SADVV	IPVV

Move

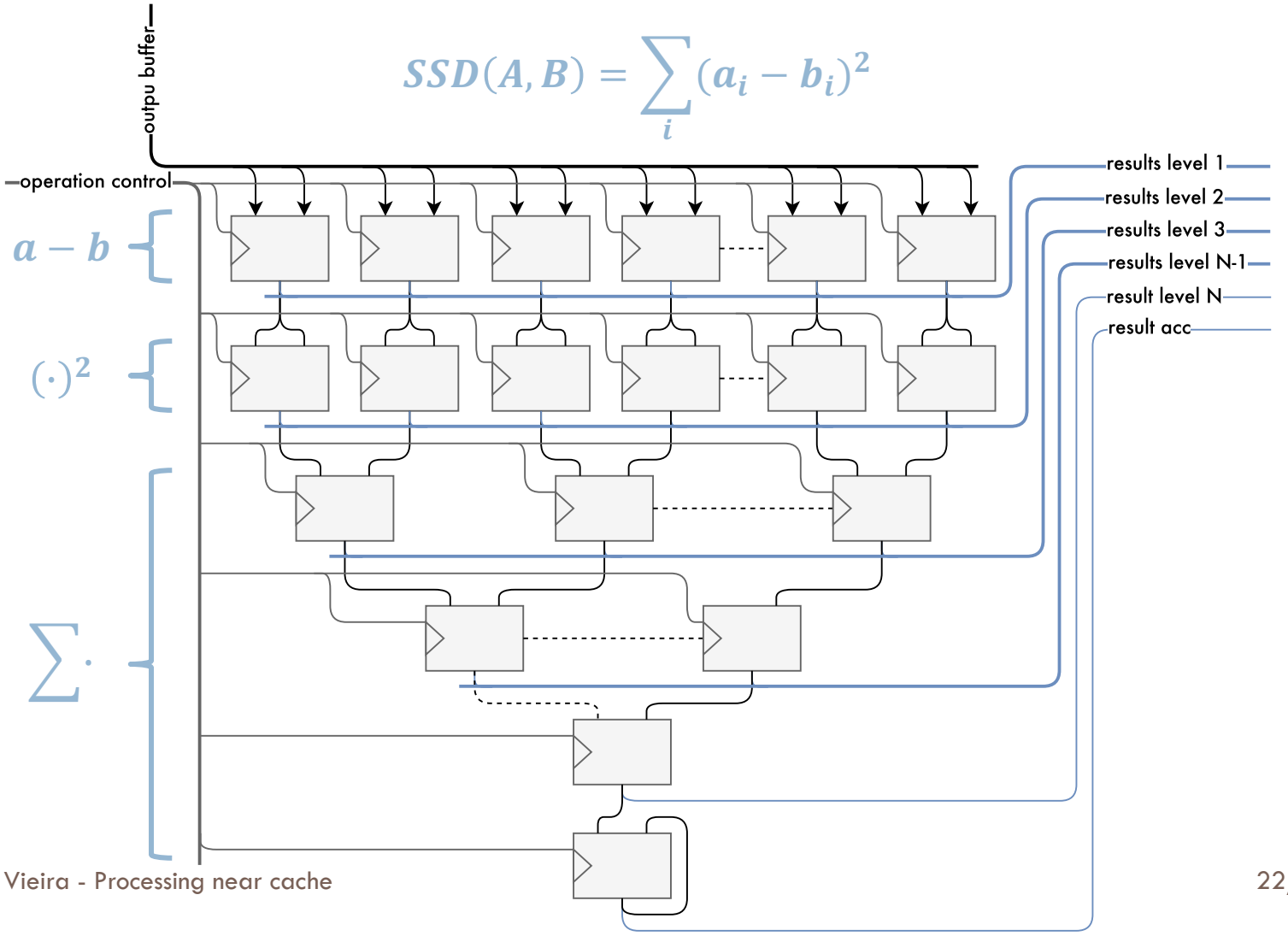
INITC	COPYV

48

Let us check one operation example

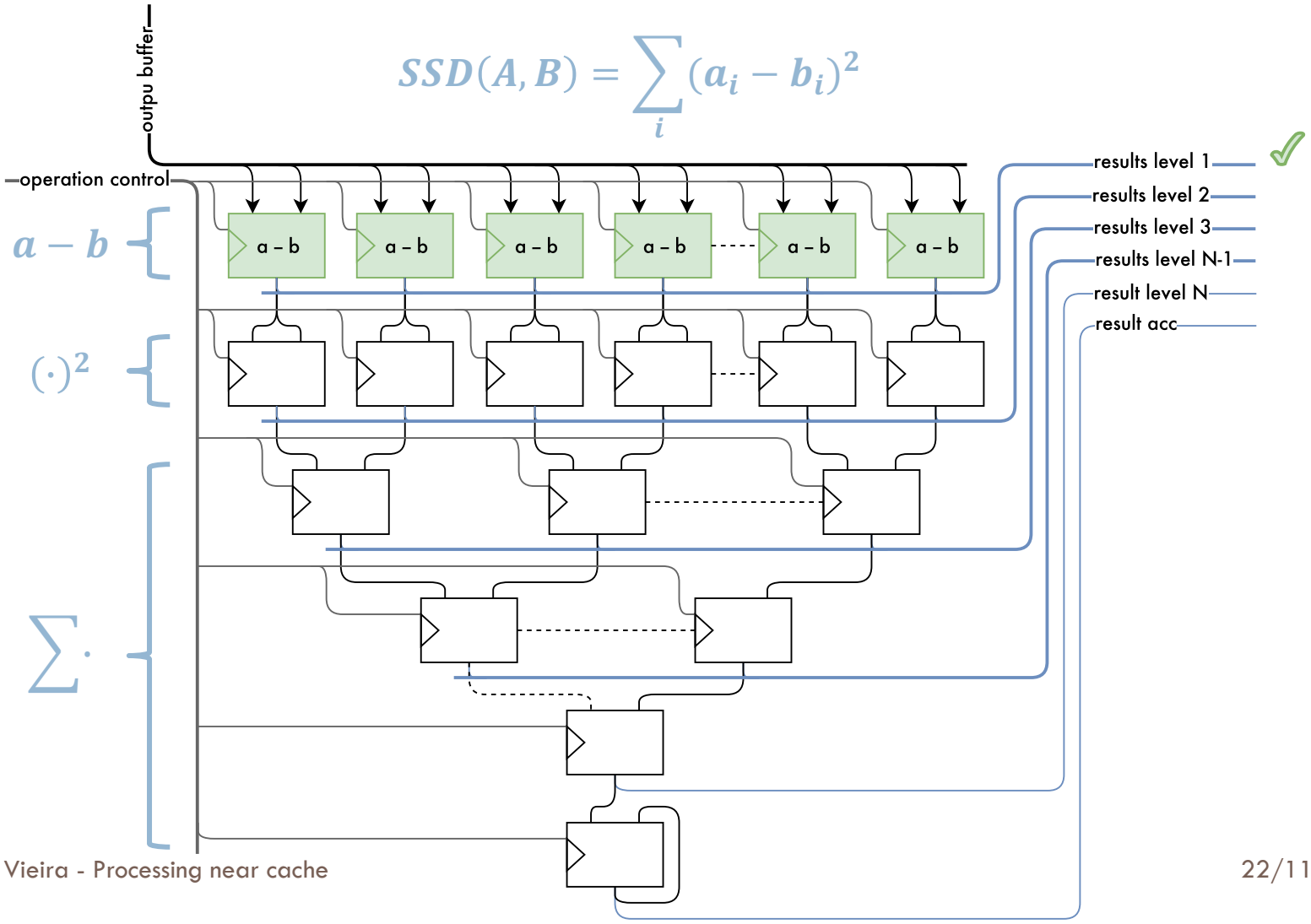
Example: sum of square differences (1)

$$SSD(A,B) = \sum_i (a_i - b_i)^2$$



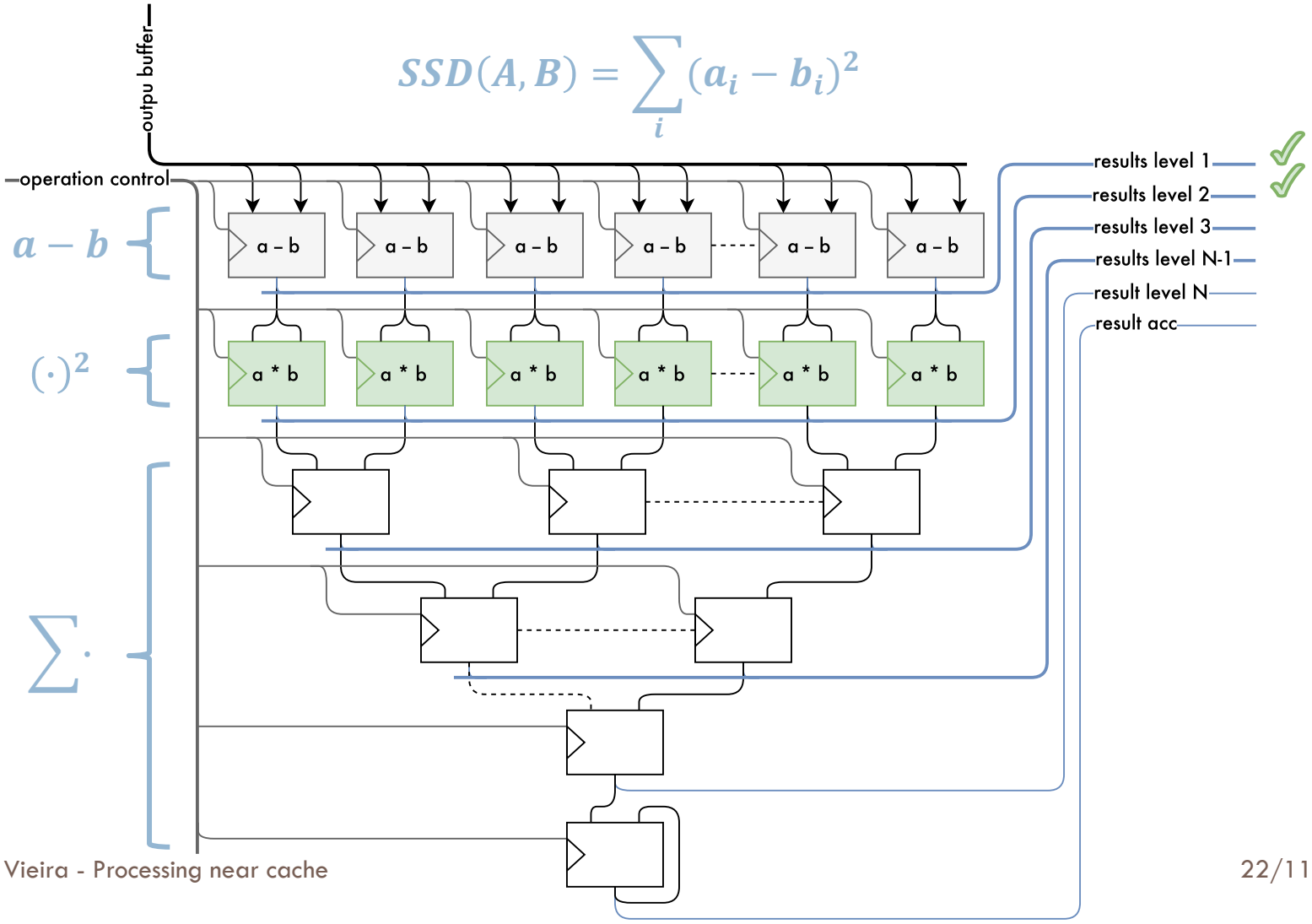
Example: sum of square differences (2)

$$SSD(A,B) = \sum_i (a_i - b_i)^2$$



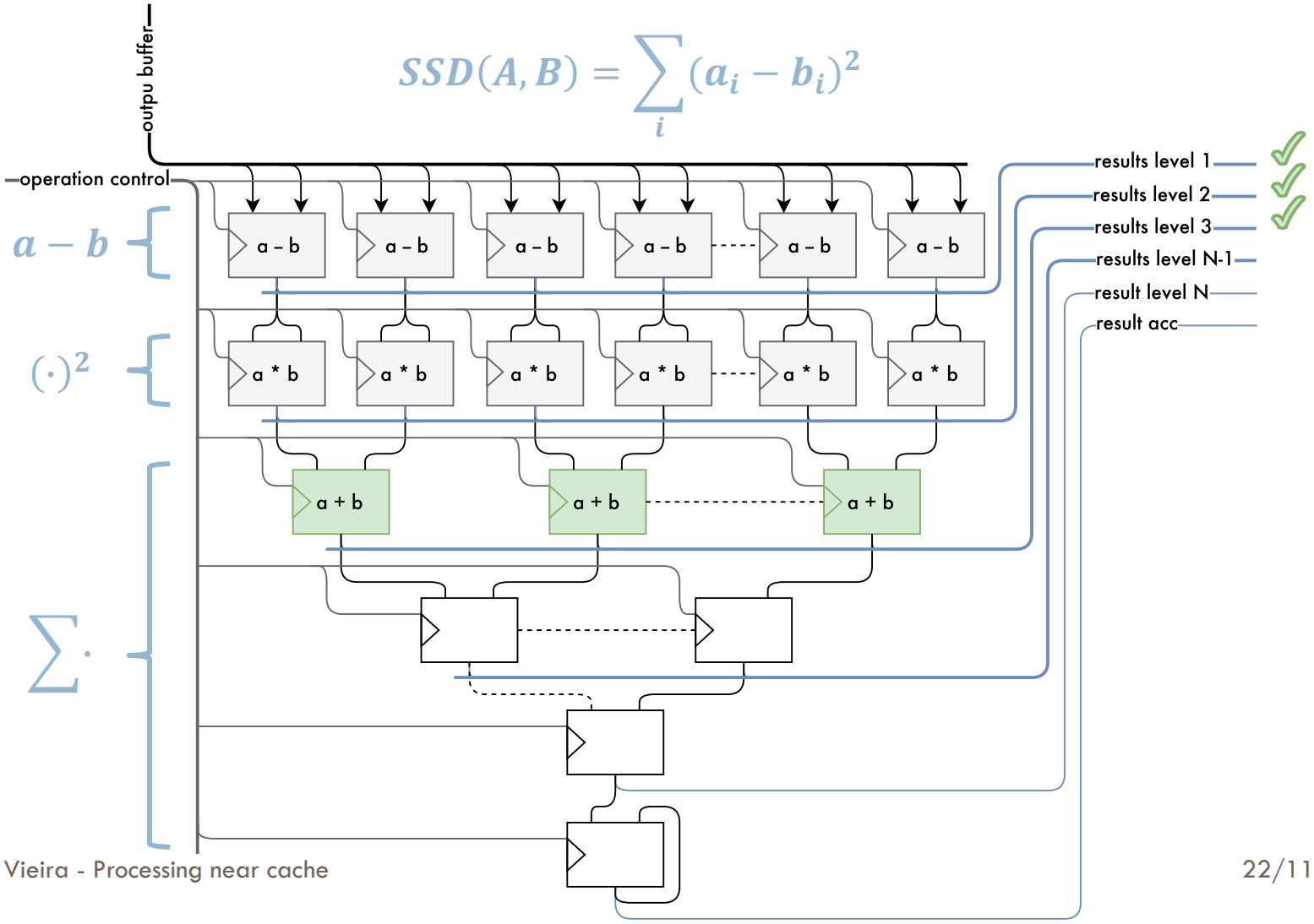
Example: sum of square differences (3)

$$SSD(A,B) = \sum_i (a_i - b_i)^2$$



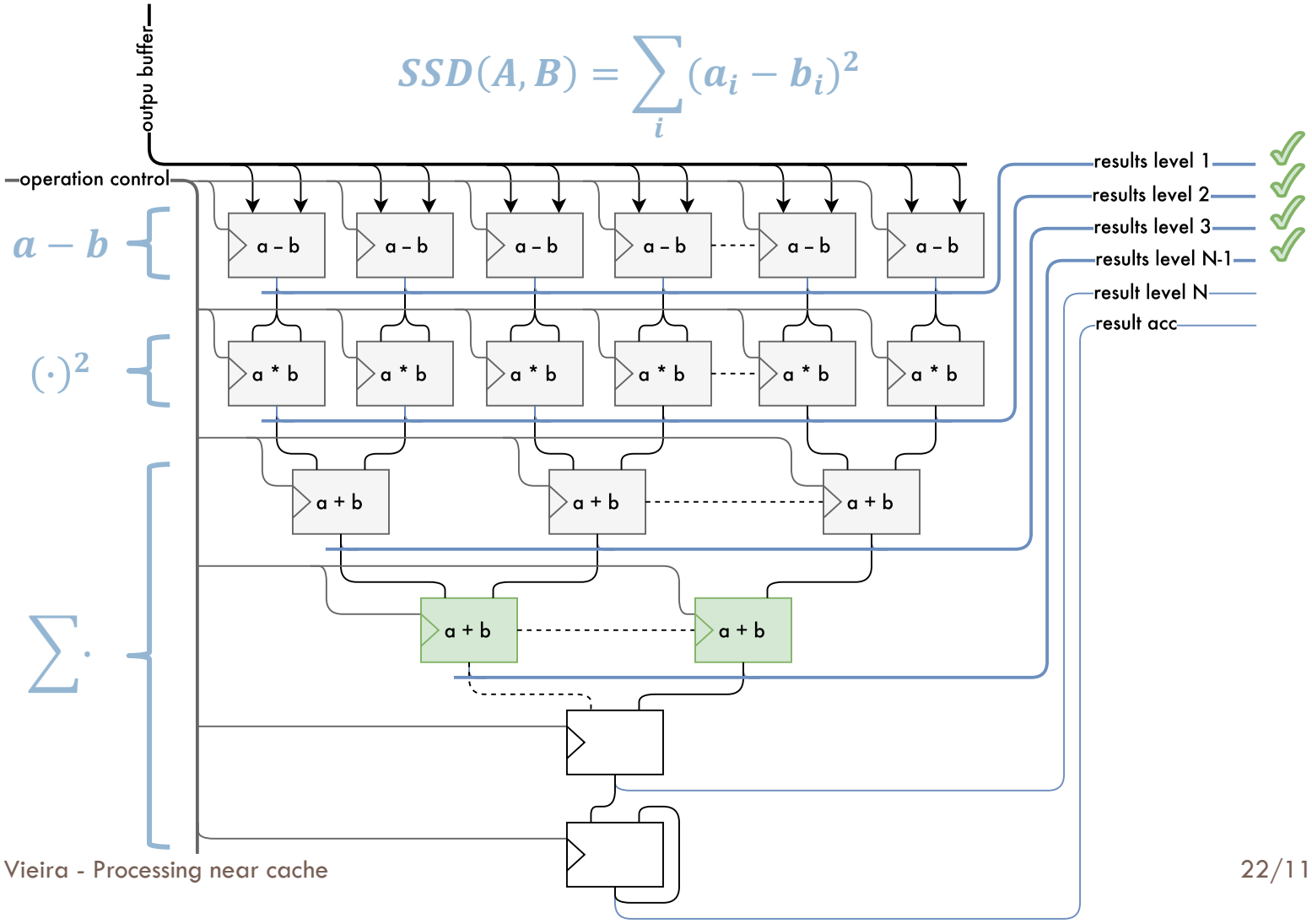
Example: sum of square differences (4)

$$SSD(A,B) = \sum_i (a_i - b_i)^2$$



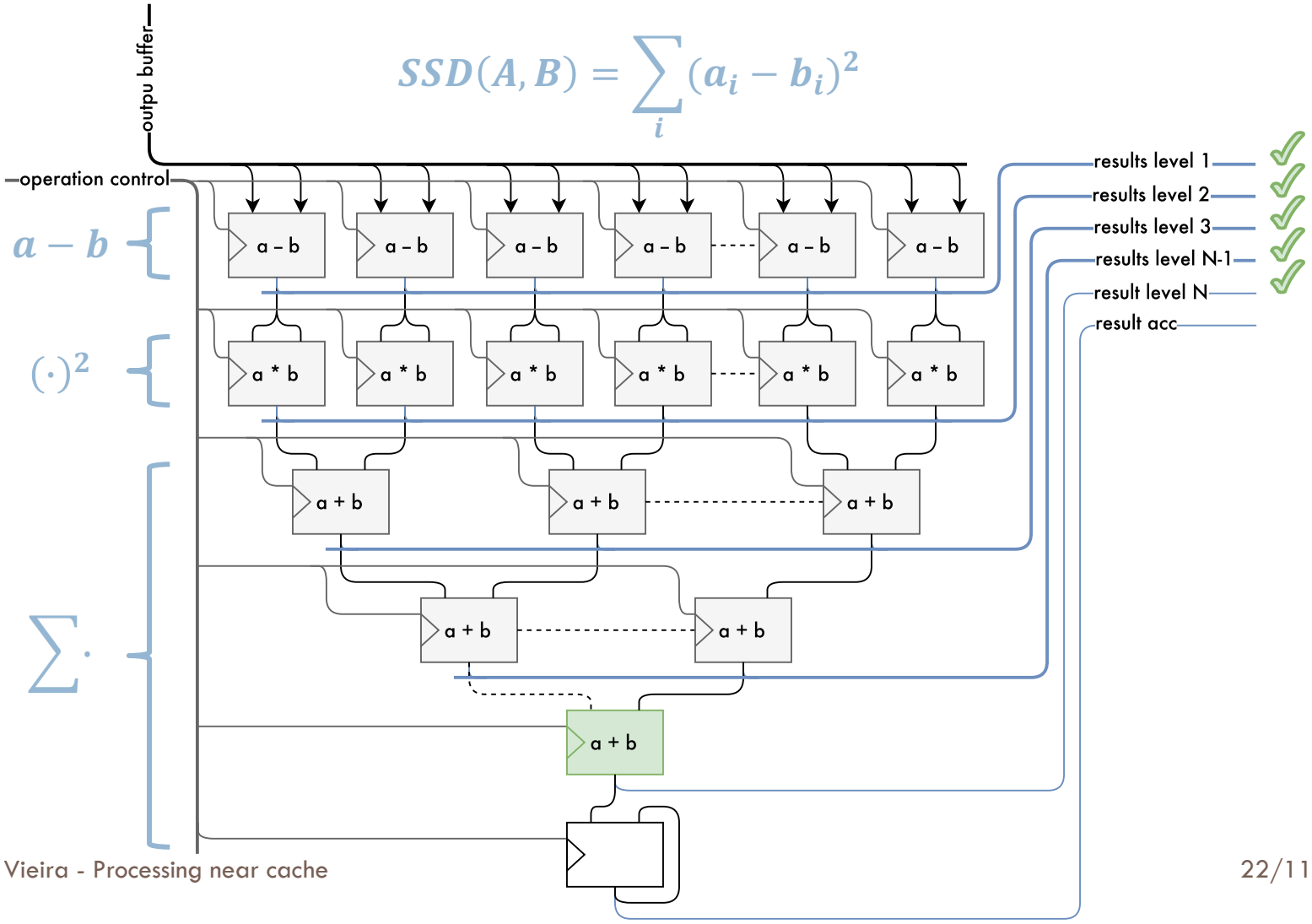
Example: sum of square differences (5)

$$SSD(A,B) = \sum_i (a_i - b_i)^2$$



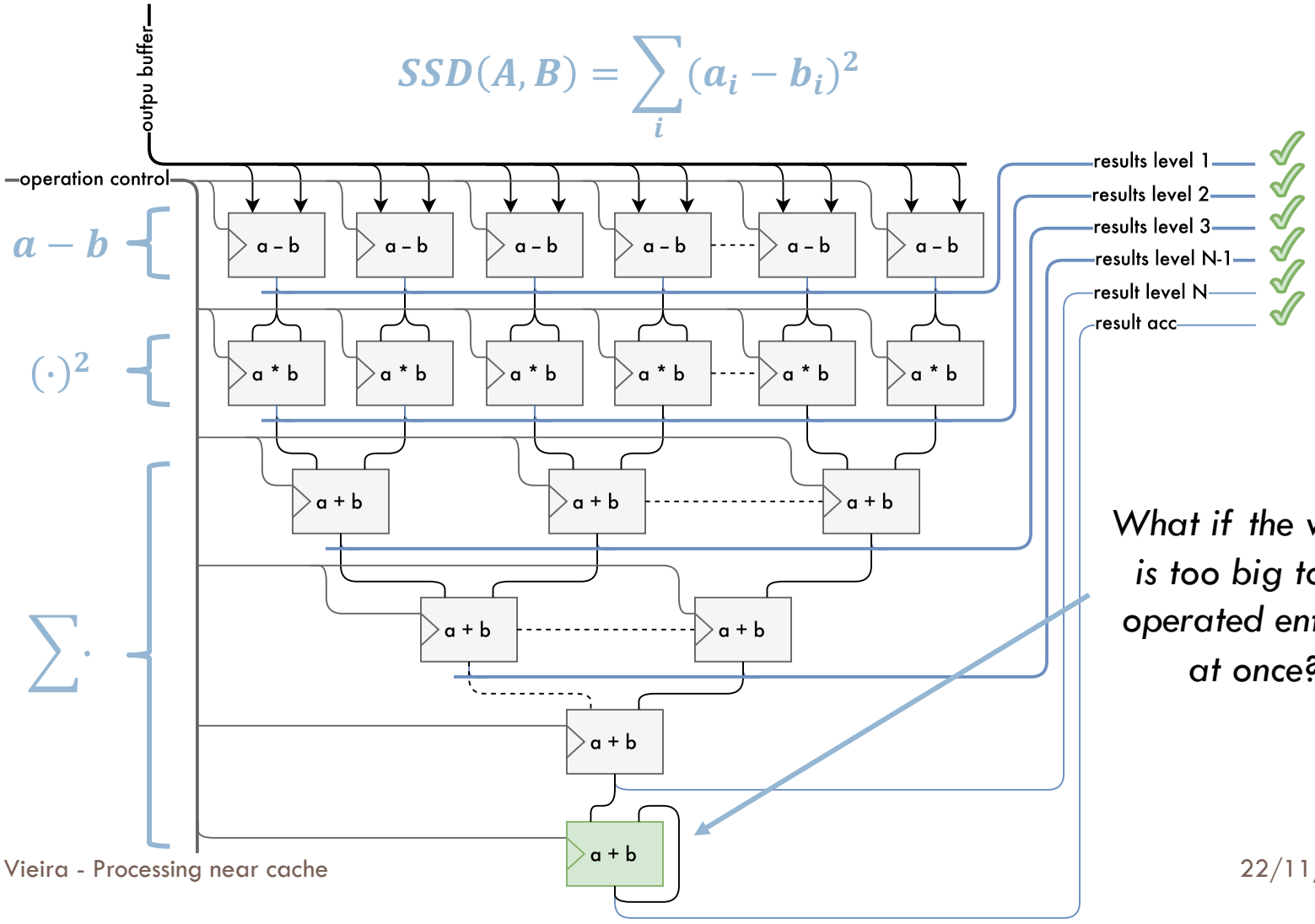
Example: sum of square differences (6)

$$SSD(A,B) = \sum_i (a_i - b_i)^2$$



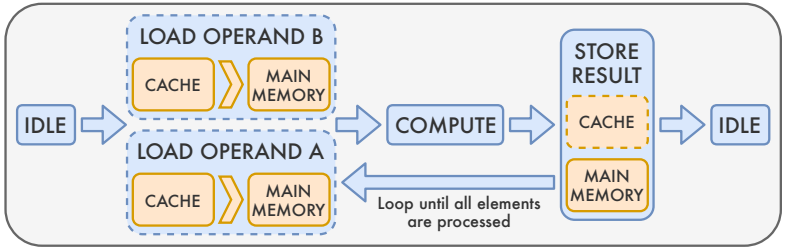
Example: sum of square differences (7)

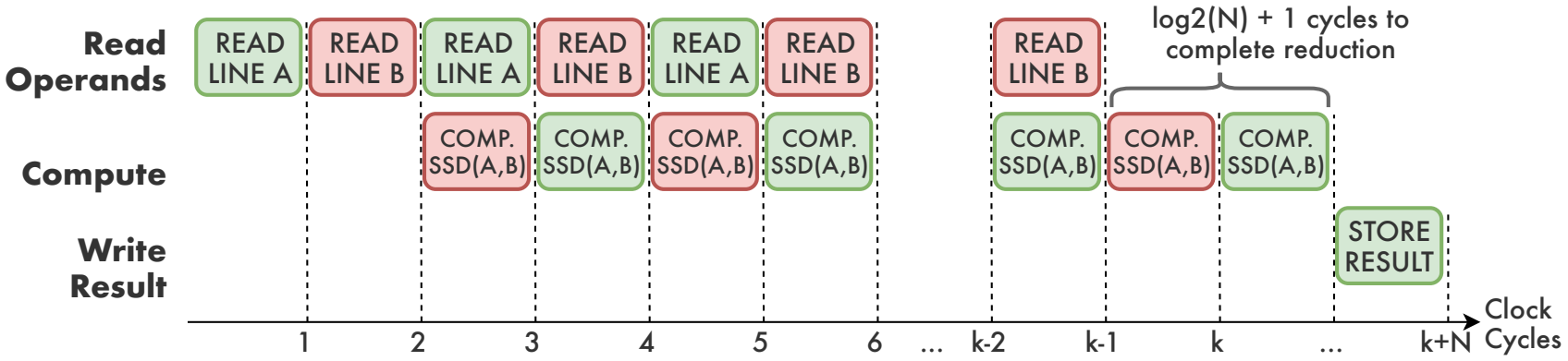
$$SSD(A,B) = \sum_i (a_i - b_i)^2$$



What if the vector is too big to be operated entirely at once?

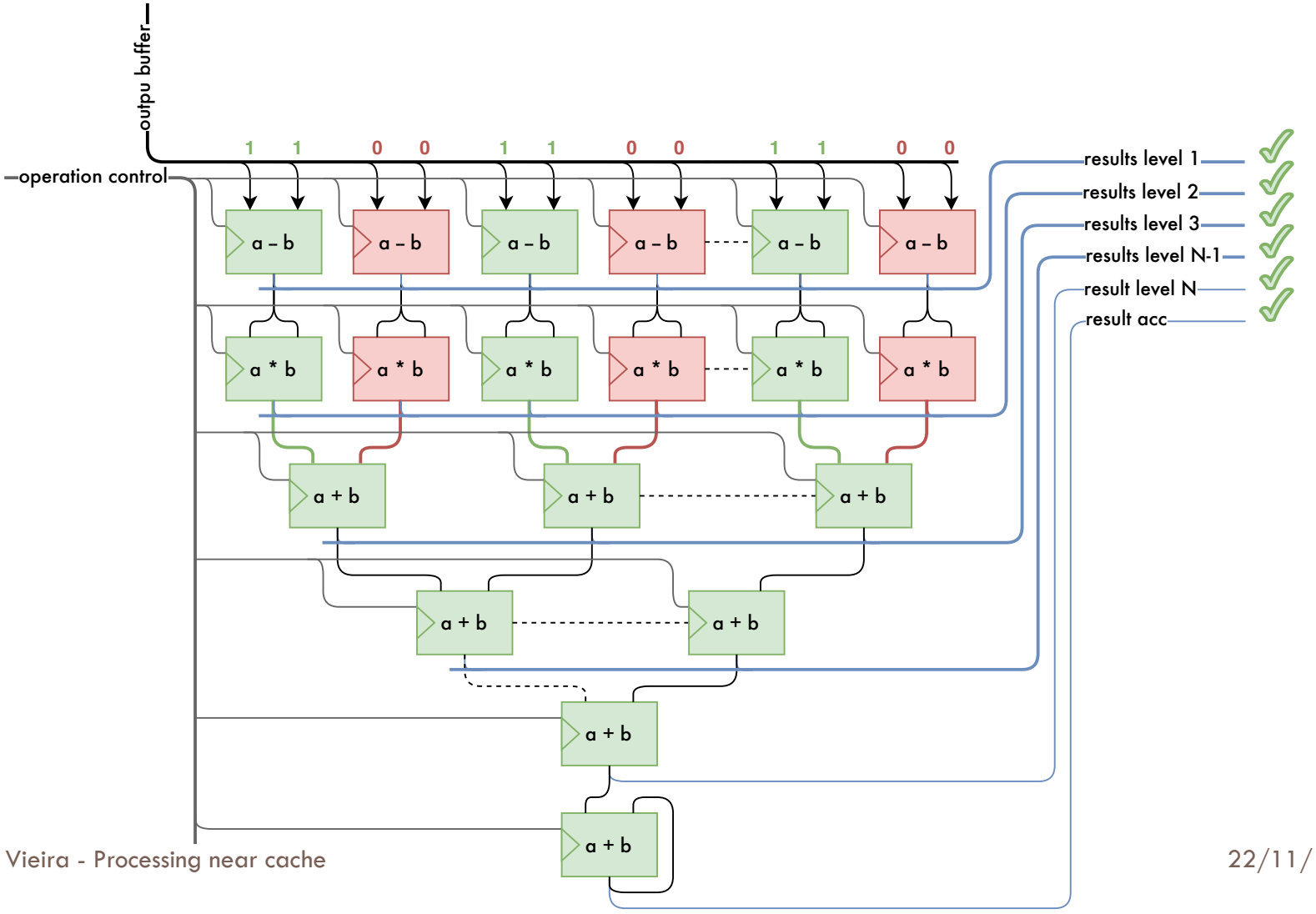
Pipeline execution/hardware loops



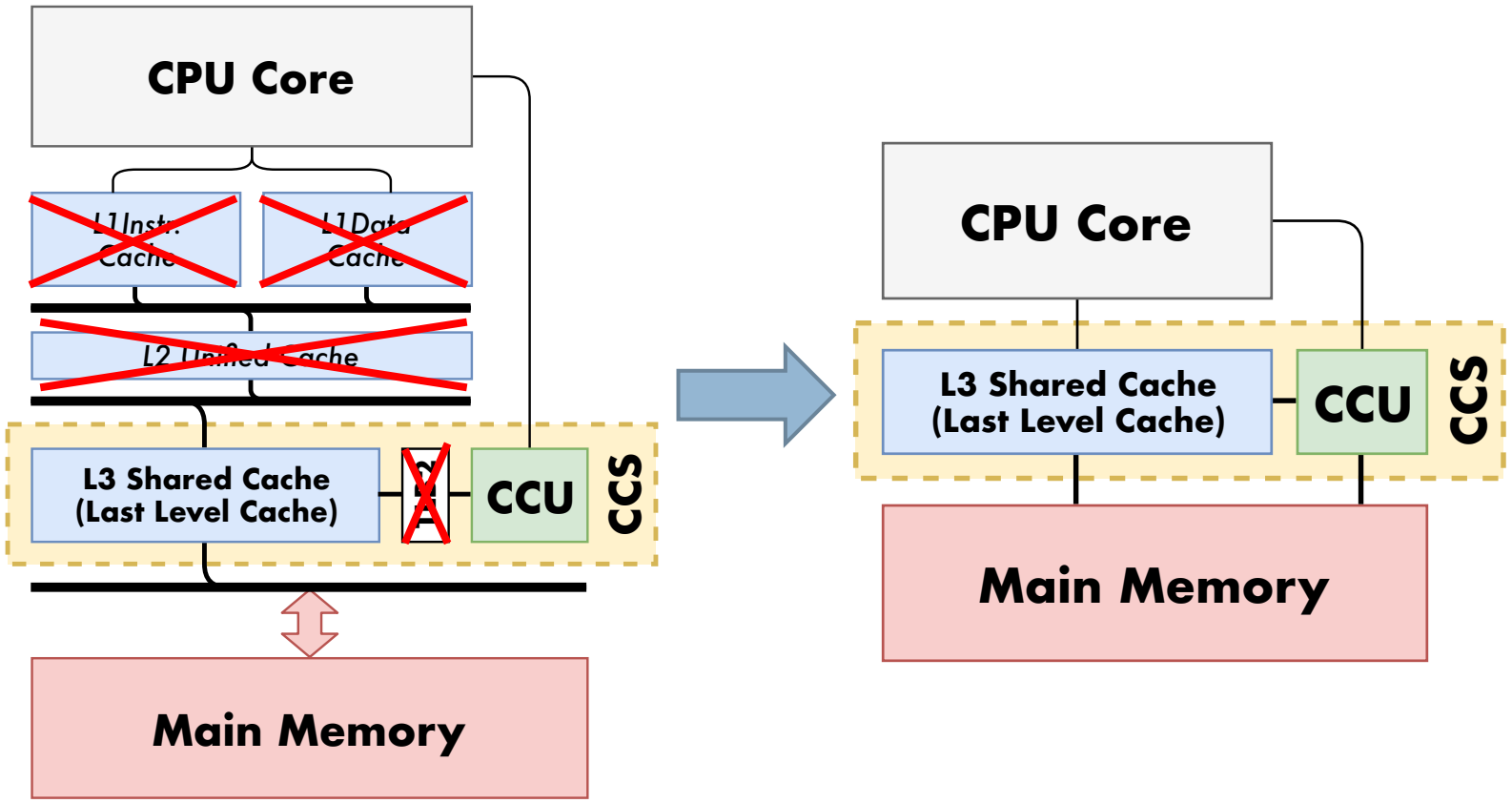


But what happens if the vector is not aligned or the data is not contiguous?

Execution mask



Implementation and testing technologies



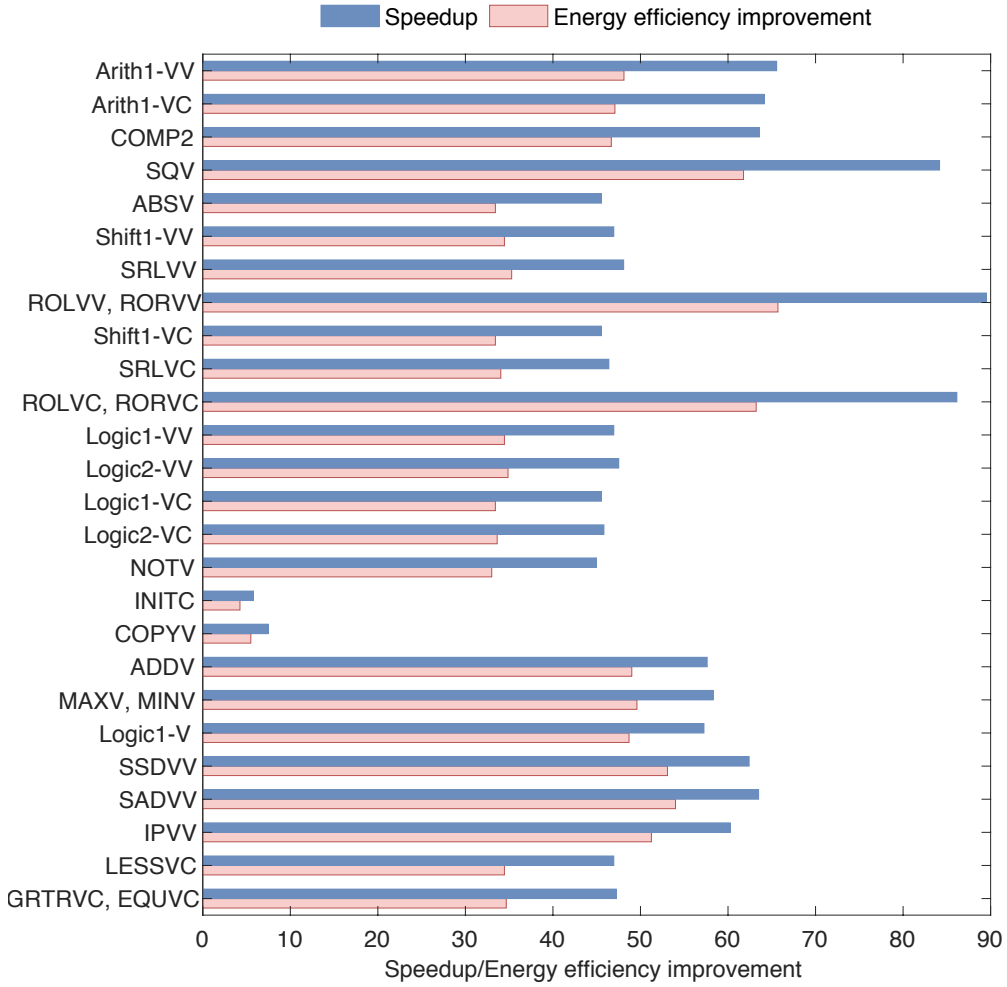
System specifications

- CPU Core: **MB-LITE soft-core;**
- **Single-level cache:**
  - 16 lines with 2048 bit per line;
  - Direct mapped.
- Main memory:
  - 64 KB capacity;
  - Byte-addressable;
  - Built with **FPGA native BRAMs.**
- **No support for virtual memory**

		CCS	MB-LITE	MB-LITE + CCS
Frequency [MHz]		100	100	67
Resources	**LUTs**	87,675	6,351	91,596
	DSPs	192	3	195
Power [W]	**Static**	0.341	0.321	0.345
	Dynamic	0.502	0.087	0.612

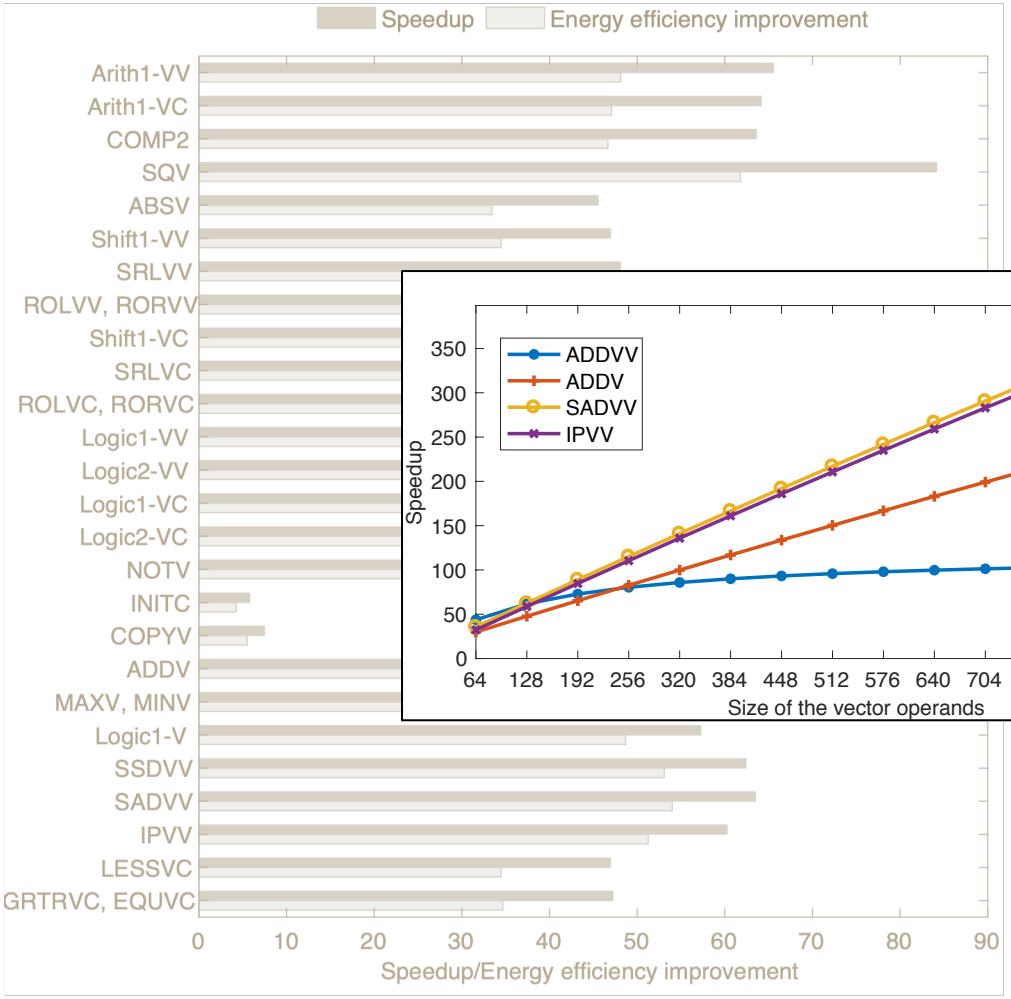
Synthetized for Xilinx Virtex-7 VC709 Development Board using Xilinx Vivado 2018.1

Experimental results



64-items	Speedup	Energy Efficiency
Max	90×	66×
Avg	53×	41×
Min	6×	4×

Speedup scales with vector size



peedup	Energy Efficiency
90×	66×
53×	41×
6×	4×

Application benchmarks

- K-Nearest Neighbors

Speedup

```
// calculate distances
for (i = 0; i < N; i++) {
  __ccs_setup(SSDVV, N, 0, b, (a + N * i), c + i, 1);
  __ccs_start();
  __ccs_wait();
}
```

68×

- Matrix Multiplication

```
for (i = 0; i < 1; i++) {
  for (j = 0; j < MATRIX_SIZE; j++) {
    __ccs_setup(IPVV, MATRIX_SIZE, 0, (mta + i * MATRIX_SIZE), (mtb + j * MATRIX_SIZE), mtc, 1);
    __ccs_start();
    __ccs_wait();
  }
}
```

54×

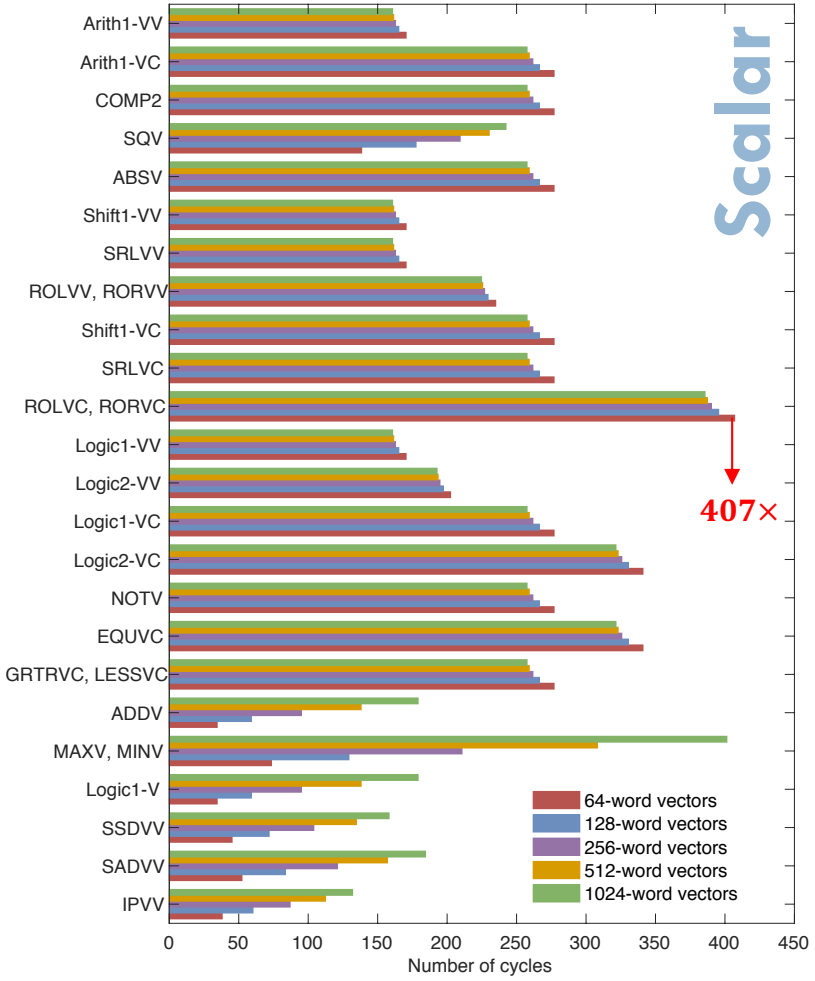
- Linear Regression

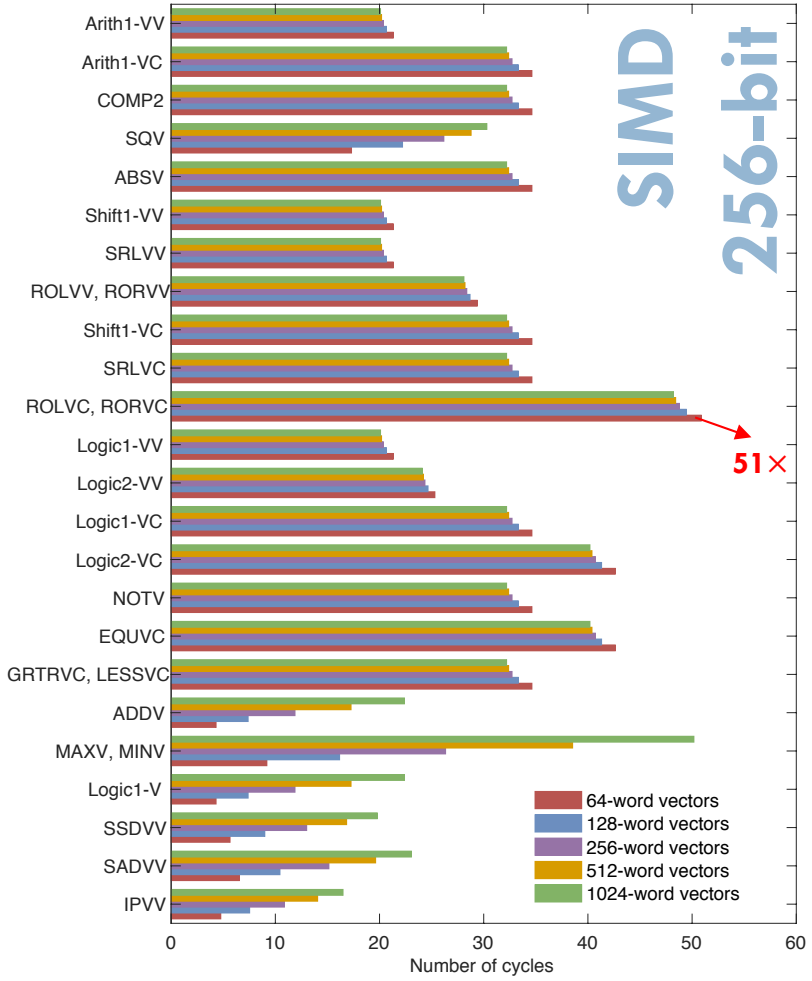
```
__ccs_setup(ADDV,  VEC_SIZE, 0, x, NULL, sum_xx,        1); __ccs_start(); __ccs_wait();
__ccs_setup(ADDV,  VEC_SIZE, 0, y, NULL, sum_yy,        1); __ccs_start(); __ccs_wait();
__ccs_setup(SQV,   VEC_SIZE, 0, x, NULL, z,             1); __ccs_start(); __ccs_wait();
__ccs_setup(ADDV,  VEC_SIZE, 0, z, NULL, sum_xx_square, 1); __ccs_start(); __ccs_wait();
__ccs_setup(MULVV, VEC_SIZE, 0, x, y,    z,             1); __ccs_start(); __ccs_wait();
__ccs_setup(ADDV,  VEC_SIZE, 0, z, NULL, sum_xy,        1); __ccs_start(); __ccs_wait();

*a = (VEC_SIZE * (*sum_xy) - (*sum_xx) * (*sum_yy)) / (VEC_SIZE * (*sum_xx_square) - (*sum_xx) * (*sum_xx));
*b = ((*sum_yy) * (*sum_xx_square) - (*sum_xx) * (*sum_xy)) / (VEC_SIZE * (*sum_xx_square) - (*sum_xx) * (*sum_xx));
```

3×

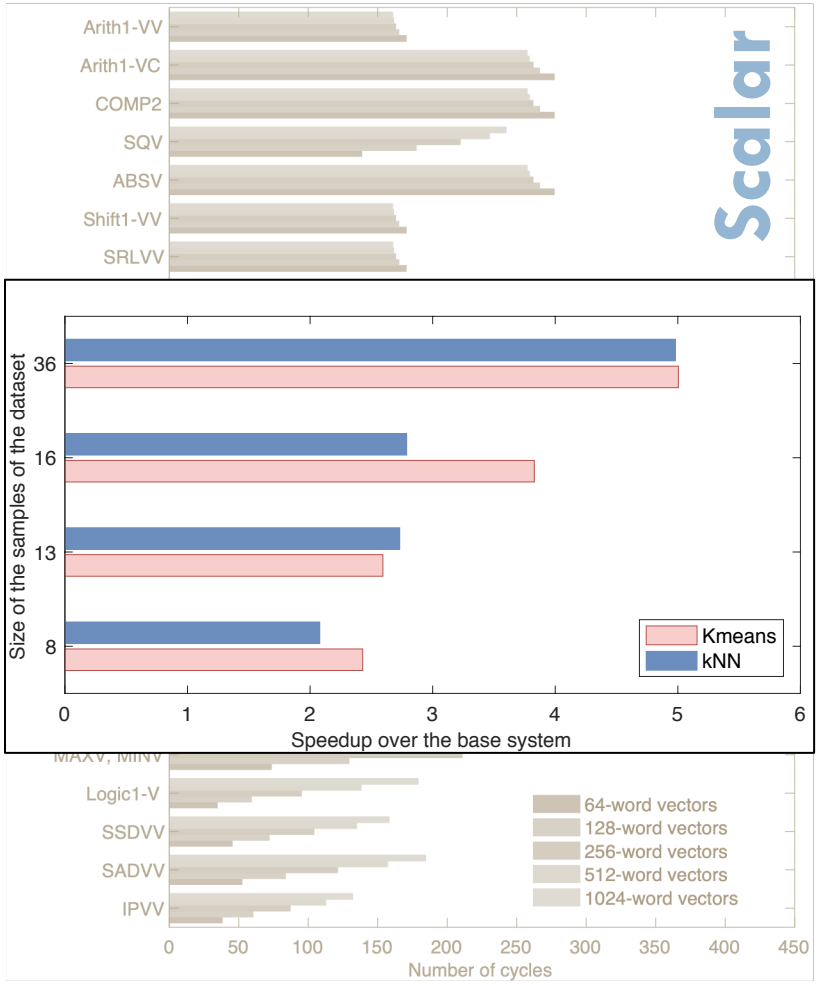
Comparison with PULPino

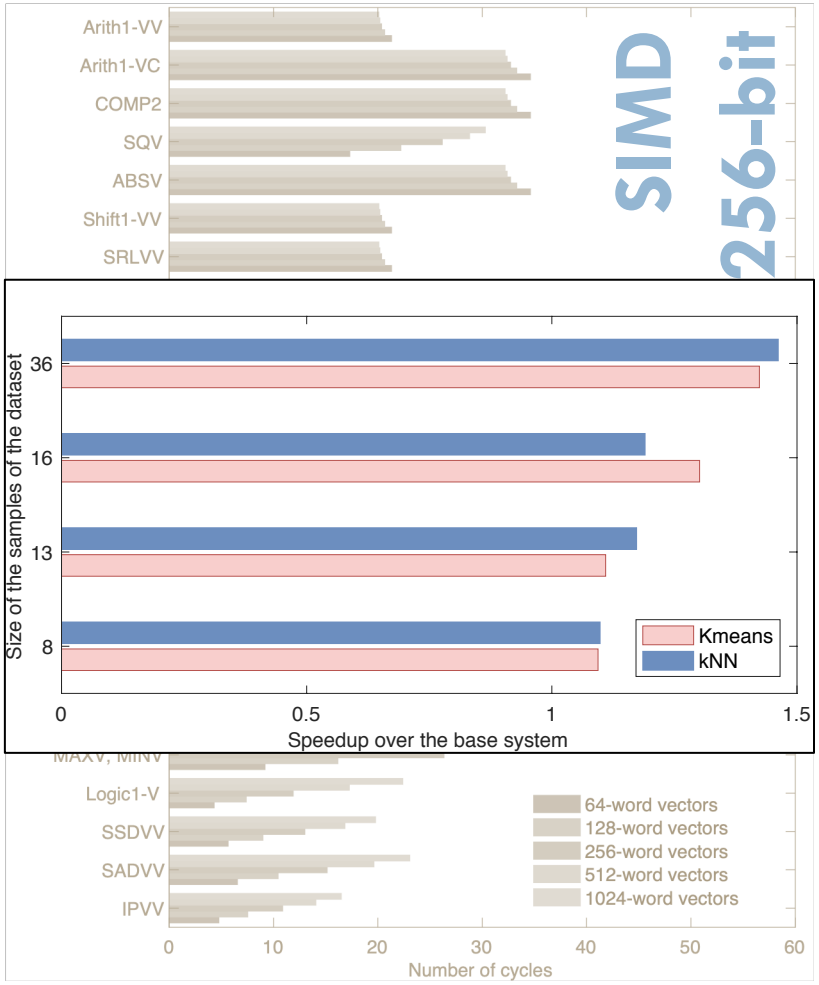




Application benchmarks for PULPino

Experimental results





Conclusions

- The **CCS brings several advantages:**
  - Data is not transferred to the core;
  - Computation is performed in parallel;
  - Cycle control overhead is totally removed;
  - Pipelined data path allows to increase performance even further.

Ongoing/Future work

- Integration with real systems and superscalars;
- Use gem5 to simulate a system featuring:
  - A superscalar (multi-)core;
  - A multi-level cache system;
  - Virtual memory support;
  - The Cache Compute System.

Speedups up to 400× and energy savings as high as 320×; 68× performance improvements for a clustering algorithm.

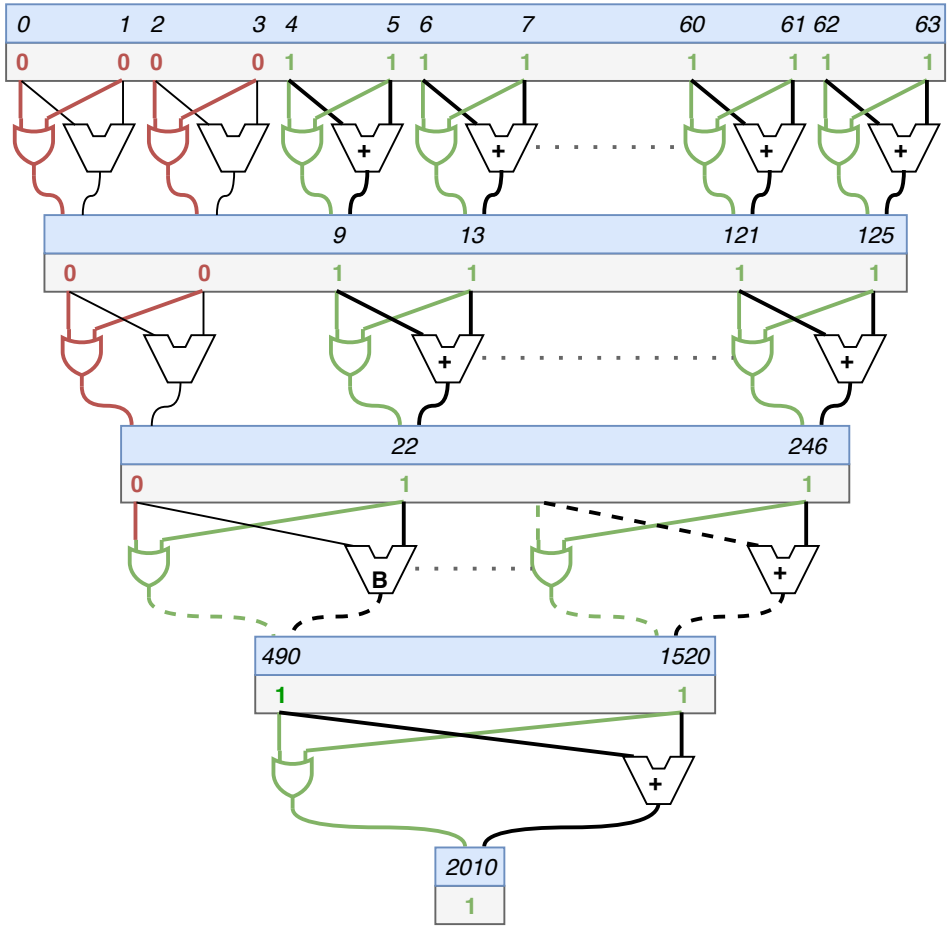
How can we do it better?

Allow programmer to create his own commands

Allow to issue entire kernels instead of commands

Overall area optimization

Execution mask generation



Hardware requirements

Xilinx Virtex-7 VC709 Development Board; Xilinx Vivado 2018.1

		CCS			MB-Lite			MB-Lite + CCS		
Frequency [MHz]		*67*	**100**		*67*	**100**		**67**		
		Utilization	**Available**	**Utilization (%)**	**Utilization**	**Available**	**Utilization (%)**	**Utilization**	**Available**	**Utilization (%)**
Resources	*LUT*	*87,675*	*433,200*	**20.24**	*6,351*	*433,200*	**1.47**	*91,596*	*433,200*	**21.14**
	LUTRAM	*7*	*174,200*	*0.00*	*2,053*	*174,200*	*1.18*	*7*	*174,200*	*0.00*
	FF	*13,014*	*866,400*	*1.50*	*2,483*	*866,400*	*0.29*	*13,368*	*866,400*	*1.54*
	BRAM	*128*	*1,470*	*8.71*	*33.5*	*1,470*	*2.28*	*161.5*	*1,470*	*10.99*
	DSP	**192**	*3,600*	*5.33*	**3**	*3,600*	*0.08*	**195**	*3,600*	*5.42*
Power [W]	*Static*	**0.341**	*0.345*		**0.328**	*0.329*		**0.345**		
	Dynamic	**0.502**	*0.793*		**0.087**	*0.15*		**0.612**		