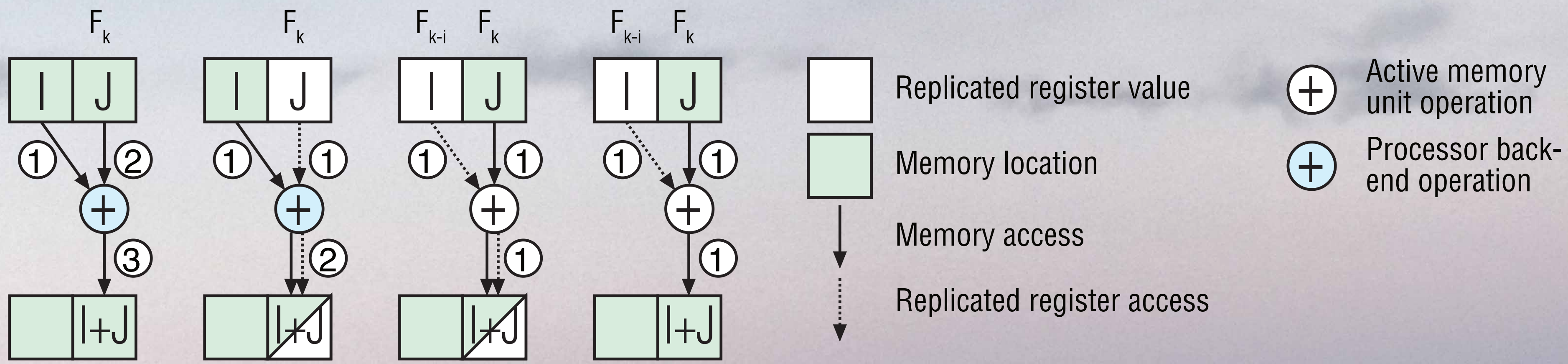
Optimizing Memory Access in TCF Processors with Compute-Update Operations



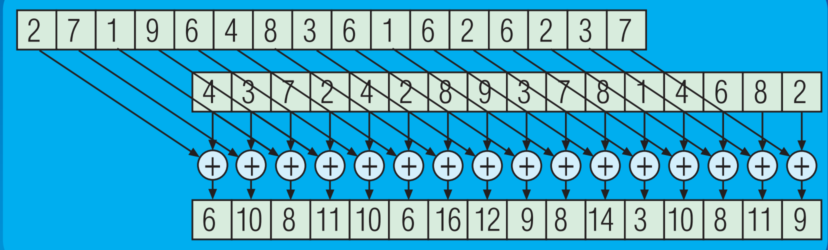
Martti Forsell VTT, Finland Martti.Forsell@VTT.Fi
Jussi Roivainen VTT, Finland
Jesper Larsson Träff TU Wien, Austria

APDCM'20 May 18, 2020, New Orleans, Louisiana, USA

VTT

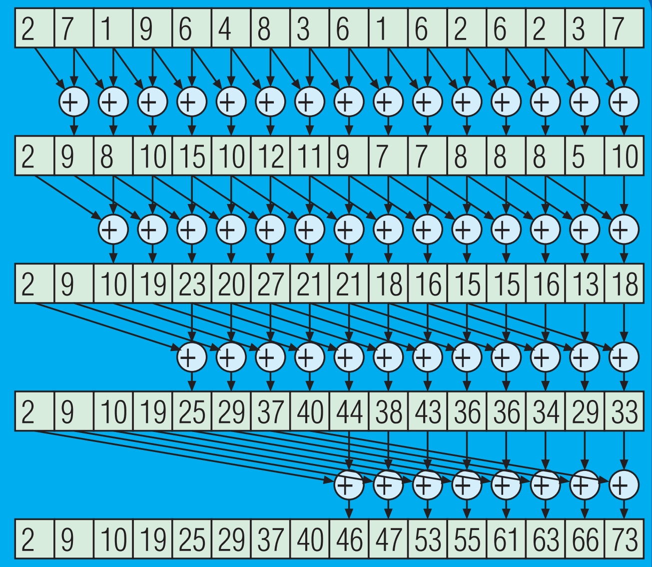
Contribution

New compute-update (CU) operations for TCF processors to optimize iterative exclusive inter-fiber memory patterns



- Accelerate matrix addition and log-prefix style patterns where multiple target locations interchange data without explicit reloads between the instructions

- Require modifications to on-chip active memory (AM) units and new CU instructions that can send their replies to another fiber than that initiating the access

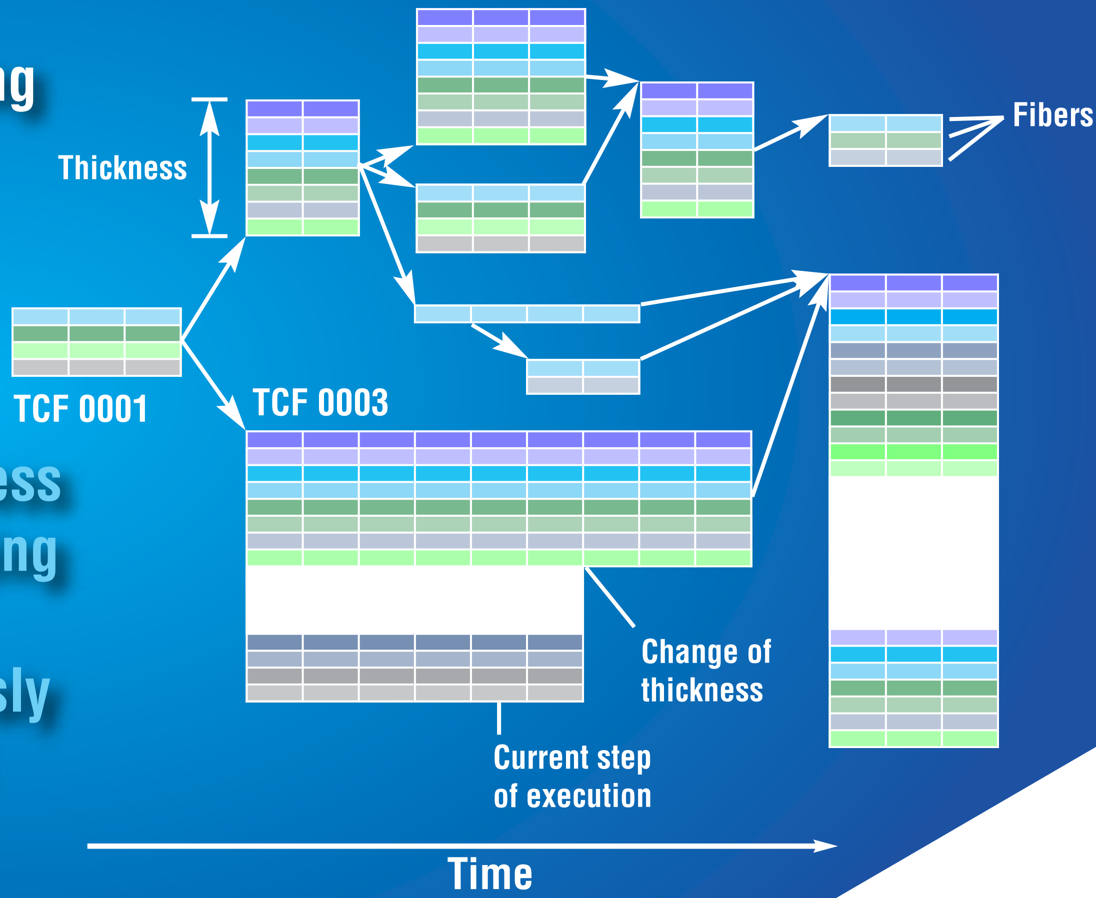


Implementation in our TPA processor with minimal HW overhead so that the expected speedups are achieved with practical functionalities

THICK CONTROL FLOWS (TCF)
[Leppänen11, Forsell13]

Combine self-similar computations flowing through the same control path into entities with a single control but multiple data paths

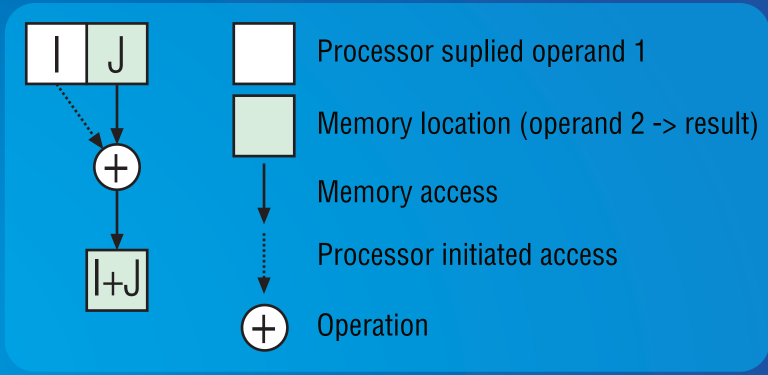
- Elements are called **fibers** to distinguish them from threads having individual control
- The number of computations is called **thickness**
- There is a mechanism for dynamically **changing the thickness**
- Fibers of a TCFs are **executed synchronously**
- Multiple TCFs can be **executed in parallel**



TCFs can also be seen as "universalized vectors" or "threads with data parallelism".

Current compute-update operations

Update the target memory location as a function of its old value and data supplied by the processor. Implement some key primitives of parallel computing:



- **Atomic instructions.** Read and conditionally change the contents of a memory location [Herlihy12].
- **Reductive multioperation instructions.** Multiple fibers concurrently reduce their data into a single value in memory [Forsell18].
- **Active memory instructions.** Memory operations employing active memory units attached to the on-chip memory modules [Forsell05, Forsell06]. Different than active memory operations (AMOs) [Fang07] and processing-in-memory (PIM) techniques [Mutlu19, Ahn15].

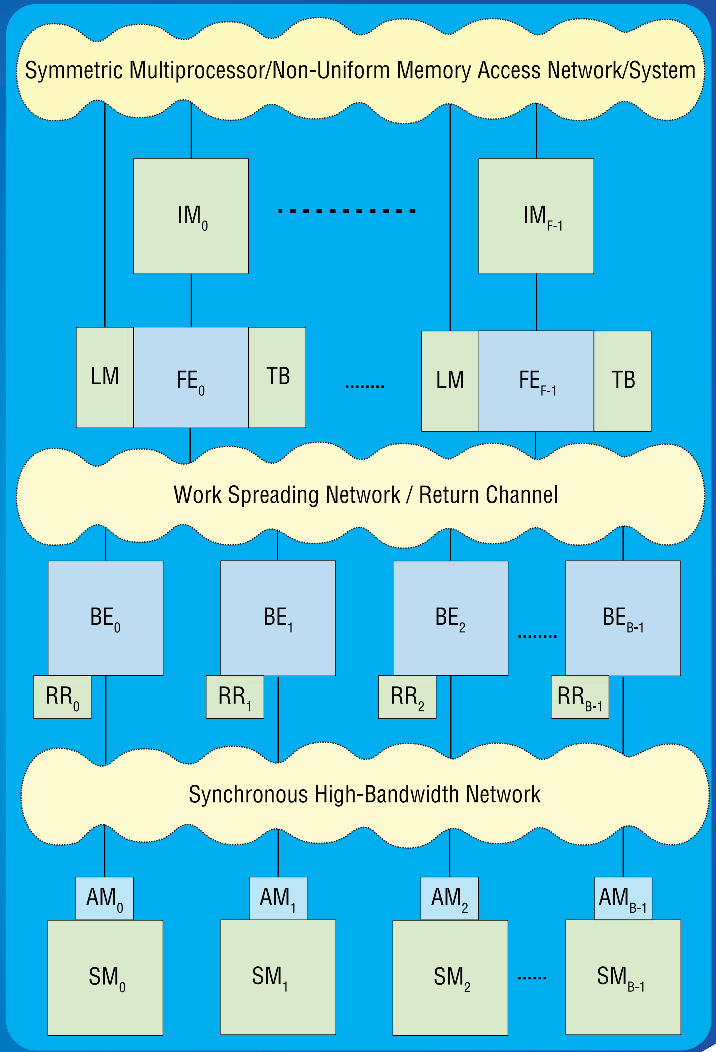
Can be used to speed up reductions and syncs. Do not work with memory access patterns in which multiple target locations interchange data.

TCF processors

Current CPUs can execute TCF programs but it is highly inefficient (slowdown 60 million) X due to slow context switching, high sync costs and OS time slicing [Forsell20].

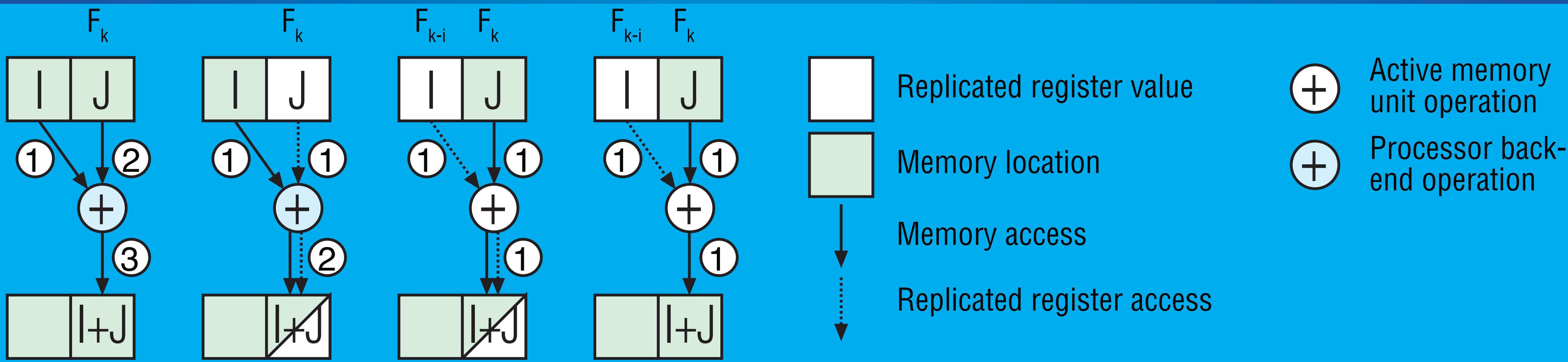
TPA is our realization of the TCF concept [Forsell16] with a number of variants [Forsell18b, Forsell18c...]:

- **F frontend (FE) PUs** for processing common parts
  - VLIW architecture with multiple parallel FUs
  - TCF buffers (TB) for holding TCFs
- **B backend (BE) PUs** for processing individual fibers
  - ESM VLIW architecture with multiple chained FUs
  - Replicated register block (RR) for fiber data
- **F instruction and local memory modules**
  - uses SMP/NUMA organization
- **B shared memory modules**
  - uses ESM organization



High-level view of TPA

New compute-update operations for optimizing memory access in TCF processors



3 accesses
1 proc. ALU op

2 accesses
1 proc. ALU op

1 access
1 AM op

1 access
1 AM op

Exclusive access MOs with inter-fiber CU ops, no reply

Inter-fiber compute-update operation

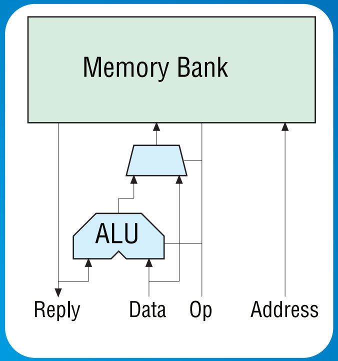
Compute-update operation: Either of the input is also the output

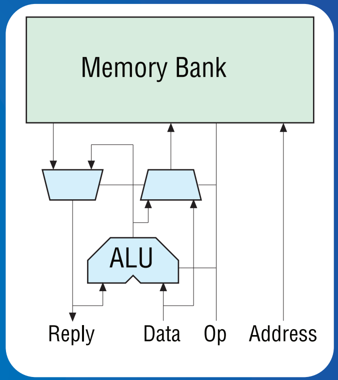
Standard two-operand compute-update operation

What is needed to implement these in TPA?

At the memory module level:

- Updated active memory units
- Support for new compute-update instructions





At the processor BE side:

- Support for new compute-update instructions
- Mechanism to annul the operations referring outside of the source array data structure defined by a base address and current thickness of the TCF
- Mechanism to determine the fiber and corresponding BE, into which the reply will be returned

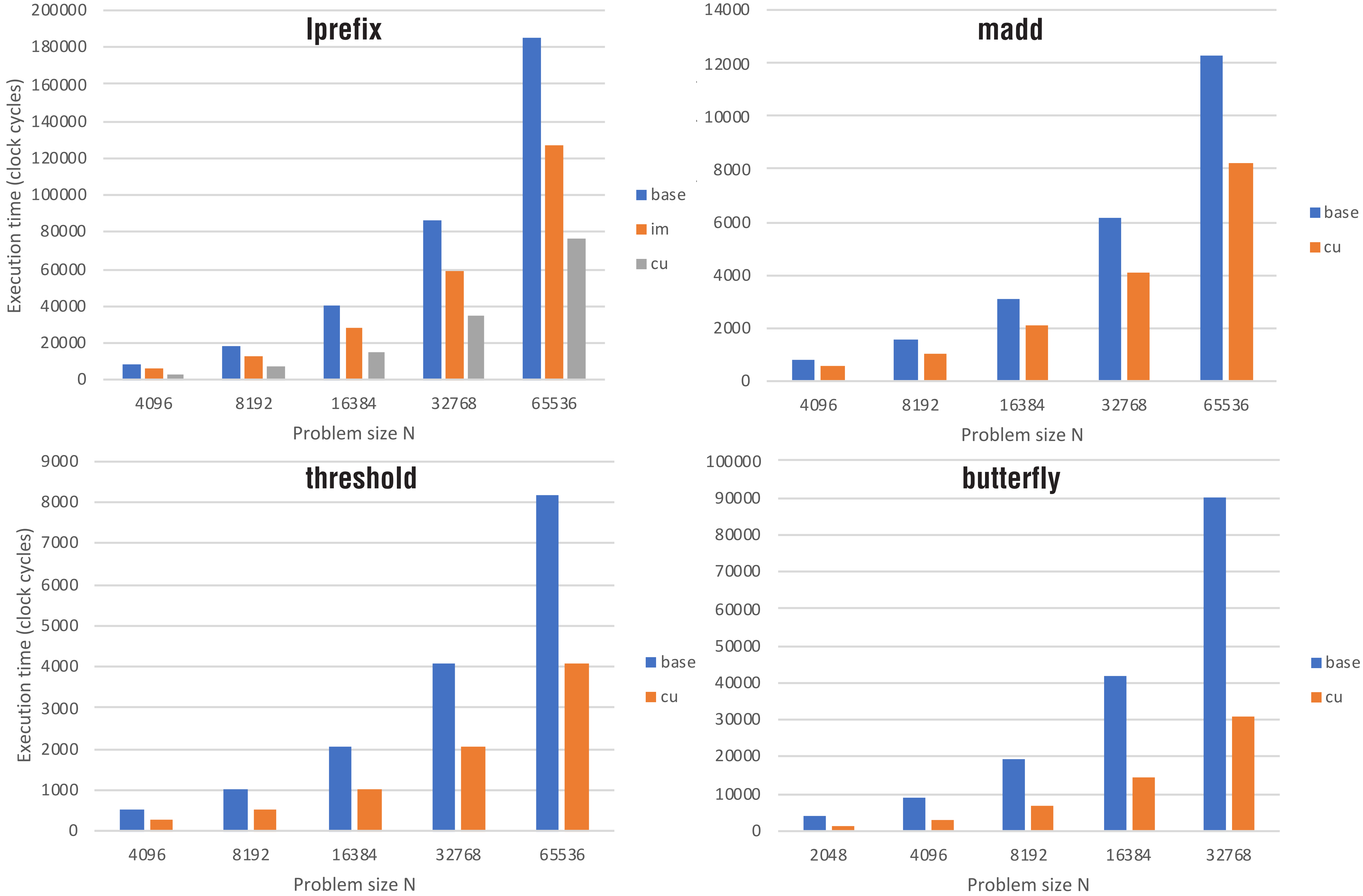
VTT

Evaluation

Processor	TPA baseline (base)	TPA interleaved mapping (im)	TPA compute-update (cu)
Processing units	1 frontend/16 backend	1 frontend/16 backend	1 frontend/16 backend
Scheme	TCF-processor	TCF-processor	TCF-processor
TCFs per frontend	128	128	128
Number of FUs	5 frontend/10 backend	5 frontend/10 backend	5 frontend/10 backend
Interconnect	4x4 mesh	4x4 mesh	4x4 mesh
Mapping of fibers to BEs	Stacked	Stacked/Interleaved	Stacked/Interleaved
Active memory units	No	No	Yes
Inter-fiber CU memory opr	No	No	Yes

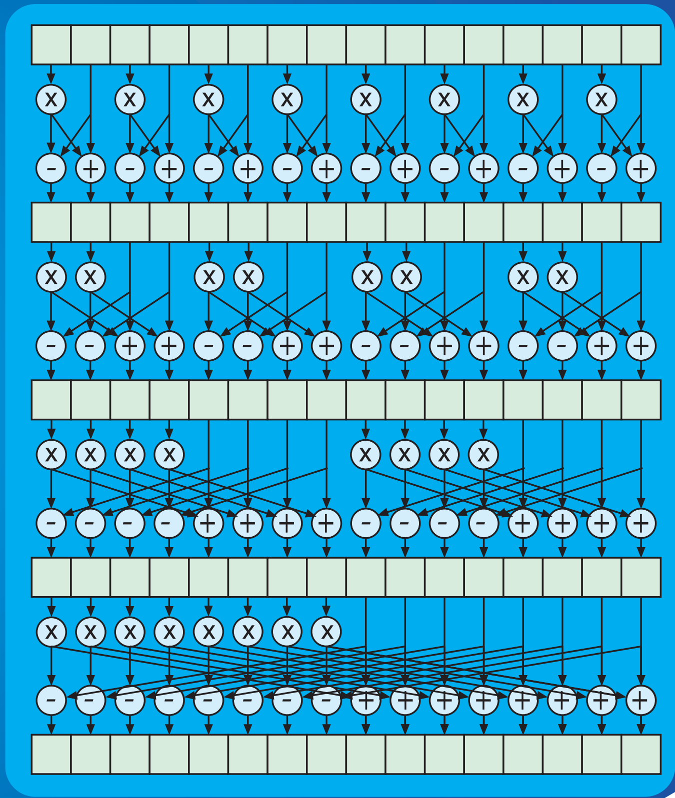
Benchmark	Description
lprefix-base	Calculates the prefix sums of an array of 4096..65536 integers using a log-prefix algorithm
lprefix-im	Calculates the prefix sums of an array of 4094..65536 integers using an optimized log-prefix algorithm
lprefix-cu	Calculates the prefix sums of an array of 4096..65536 integers using the compute-update log-prefix algorithm
madd-base	Calculates the sum of two arrays of 4096..65536 integers
madd-cu	Calculates the sum of two arrays of 4096..65536 integers
threshold-base	Applies a threshold filter to an array of 4096..65536 integers
threshold-cu	Applies a threshold filter to an array of 4096..65536 integers
butterfly-base	Calculates the entirely real-valued fft butterfly without multiplication with sine/cosine coeffs and Q-branch negation
butterfly-cu	Calculates the entirely real-valued fft butterfly without multiplication with sine/cosine coeffs and Q-branch negation

TPA baseline can execute **base** programs
TPA interleaved mapping can execute **base** and **im** programs
TPA compute-update can execute **base**, **im** and **cu** programs


lprefix
madd
threshold
butterfly
Execution time (clock cycles)
Problem size N
base
im
cu
0
20000
40000
60000
80000
100000
120000
140000
160000
180000
200000
2000
4000
6000
8000
10000
12000
14000
1000
3000
5000
7000
9000
10000
30000
50000
70000
90000
2048
4096
8192
16384
32768
65536

Conclusions

We have proposed an **architectural solution to optimize memory access in TCF processors by supporting inter-fiber CU operations**. It is based on modified AM units and special instructions that can send their reply value to another fiber than that initiating the access.



- Applies for **exclusive matrix-addition and log-prefix -style memory access patterns**.
- In comparison to the baseline TPA the speedup is
  - 150% in **log-prefix** algorithm
  - over 190% in **fft-style butterfly** algorithm
  - •50-100% in **matrix addition** and **threshold filtering**
- The **HW overhead is very low** in our silicon area and power consumption estimations

VTT