SRAM-Based MIMD AI-Accelerators for Sequence Alignment: Using the Graphcore IPU for High-Throughput Bioinformatics

Luk Burchard
Simula Research Laboratory, Norway
luk@simula.no

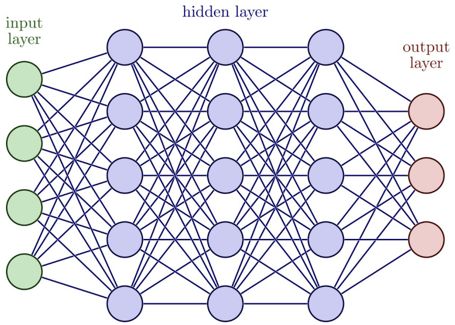
In collaboration with:
Aydın Buluç, Xing Cai, Giulia Guidi, Johannes Langguth, Max Zhao

Apr 14, 2023

GPUs started as a product for gamers, but are a great tool for accelerating scientific calculations

Image Source: Peter Langfelder, Bin Zhang, Steve Horvath; Izaak Neutelings; Meet Scott Braun, NASA, Andrey Matveev

GPUs started as a product for gamers, but are a great tool for accelerating scientific calculations



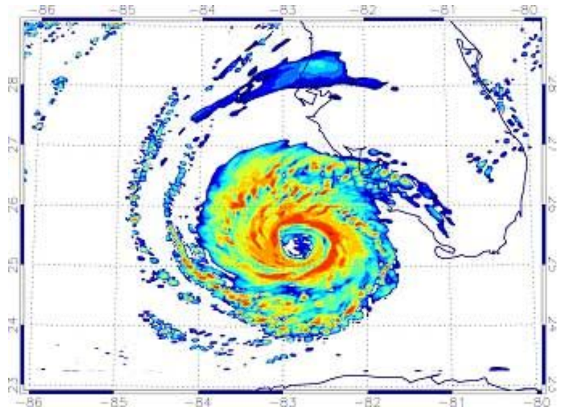
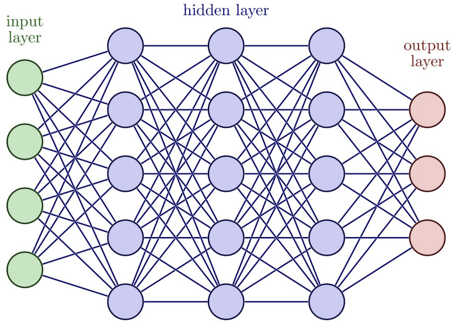
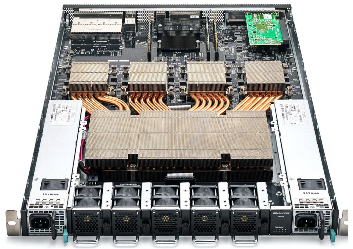
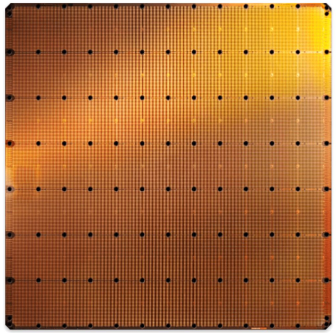


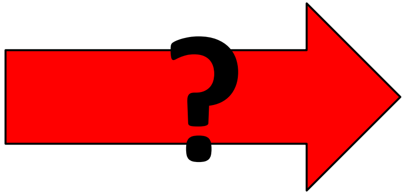
Image Source: Peter Langfelder, Bin Zhang, Steve Horvath; Izaak Neutelings; Meet Scott Braun, NASA, Andrey Matveev

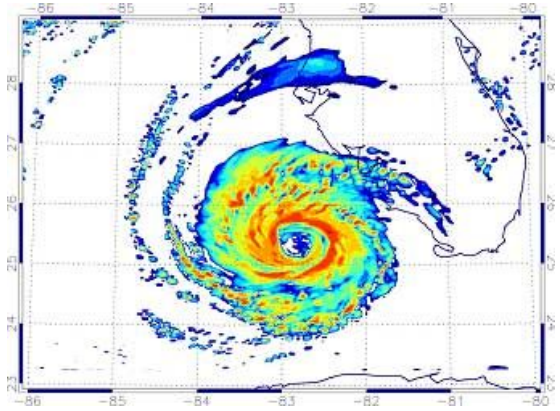
We have new hardware for AI/ML, but can they be repurposed like (GP)GPUs.











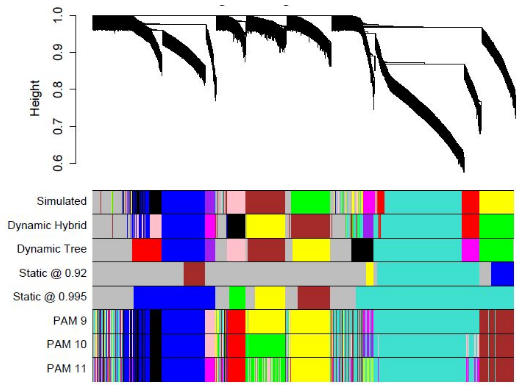


Image Source: Peter Langfelder, Bin Zhang, Steve Horvath; Izaak Neutelings; Meet Scott Braun, NASA, Andrey Matveev

We take a look at the Graphcore IPU

Relevant IPU features:

MIMD rather than SIMD

- ❑ **1472 individual cores (tiles)**

Dark silicon is SRAM

- ❑ **918MB cache**

Low memory latency

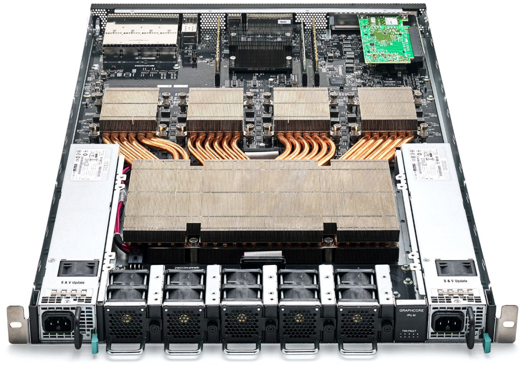
- ❑ **1 cycle each access (128 bit)**

***High* on-chip memory bandwidth**

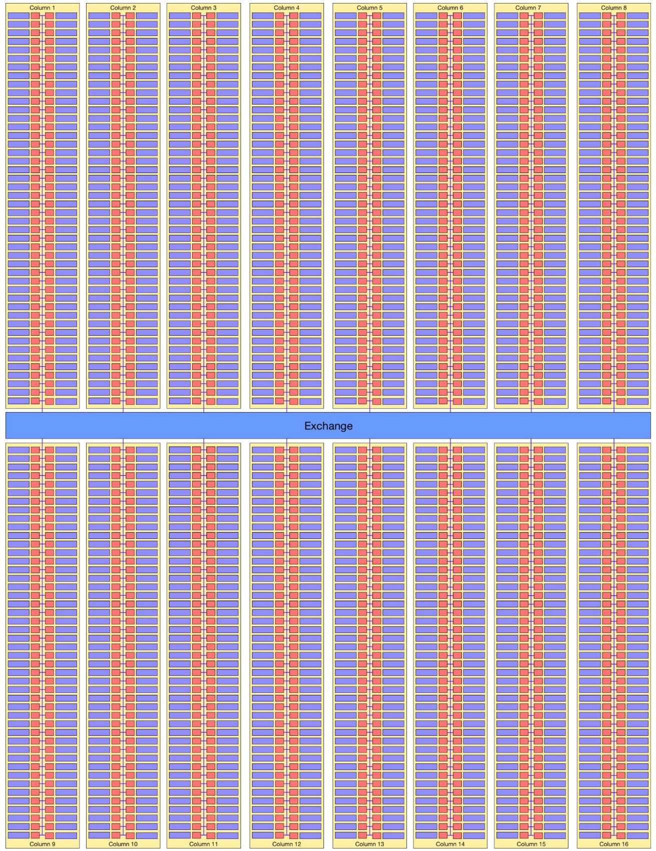
- ❑ **8 TB/s core-core**

Build for AI acceleration

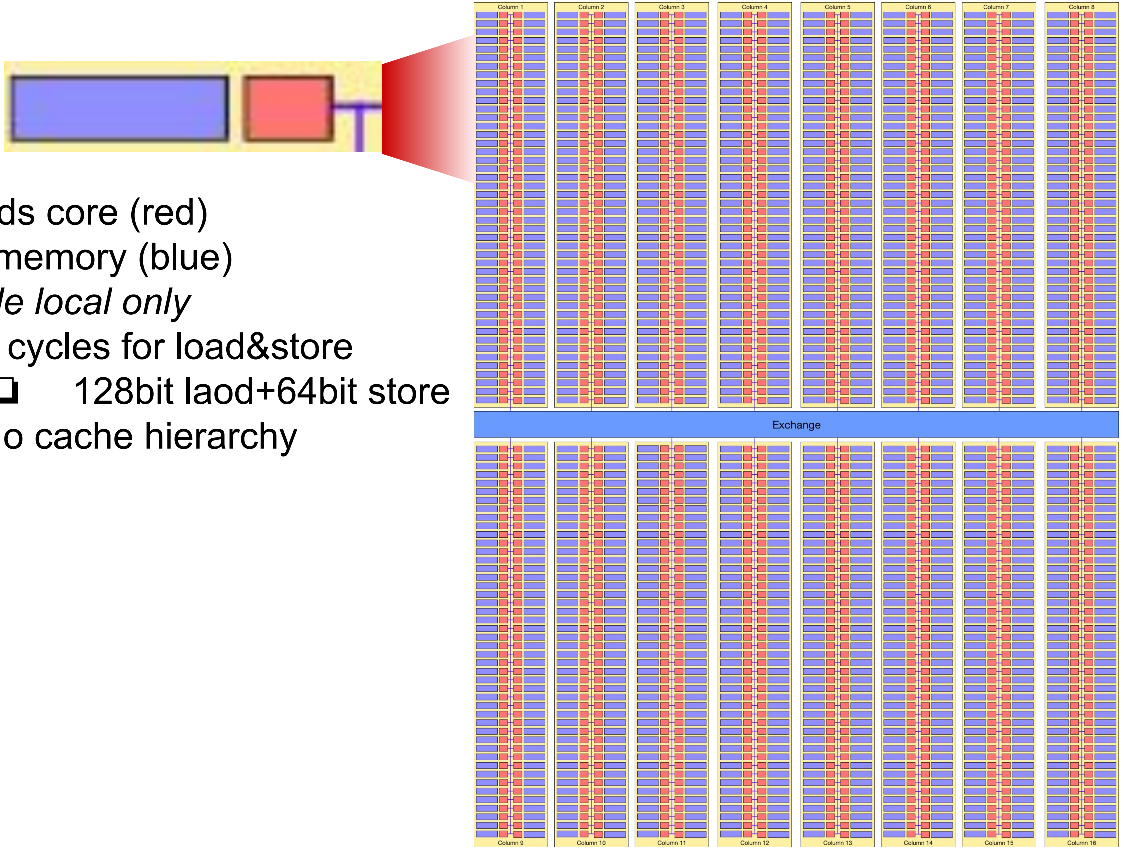
No external memory (RAM)



The IPU chip has 1472 individual cores with individual memory



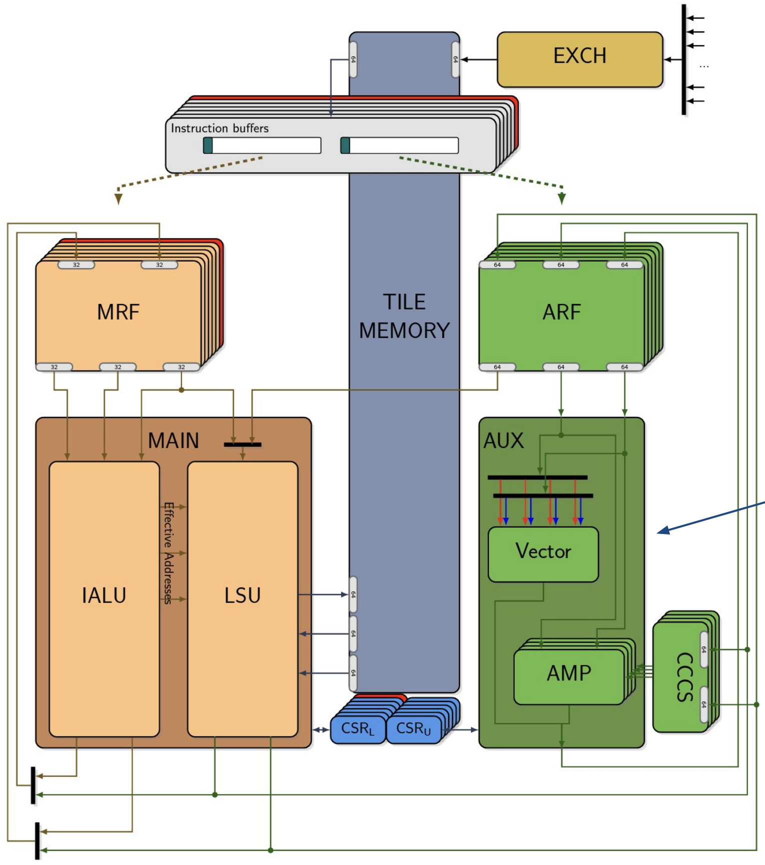
The IPU chip has 1472 individual cores and 8832 threads



6 threads core (red)
624kb memory (blue)

- *tile local only*
- 1 cycles for load&store
  - 128bit laod+64bit store
- No cache hierarchy

The ISA uses VLIW[†] for the MAIN, and AUX pipeline



Floating point unit, special AXPY instructions.

Very AI/ML workload centric.

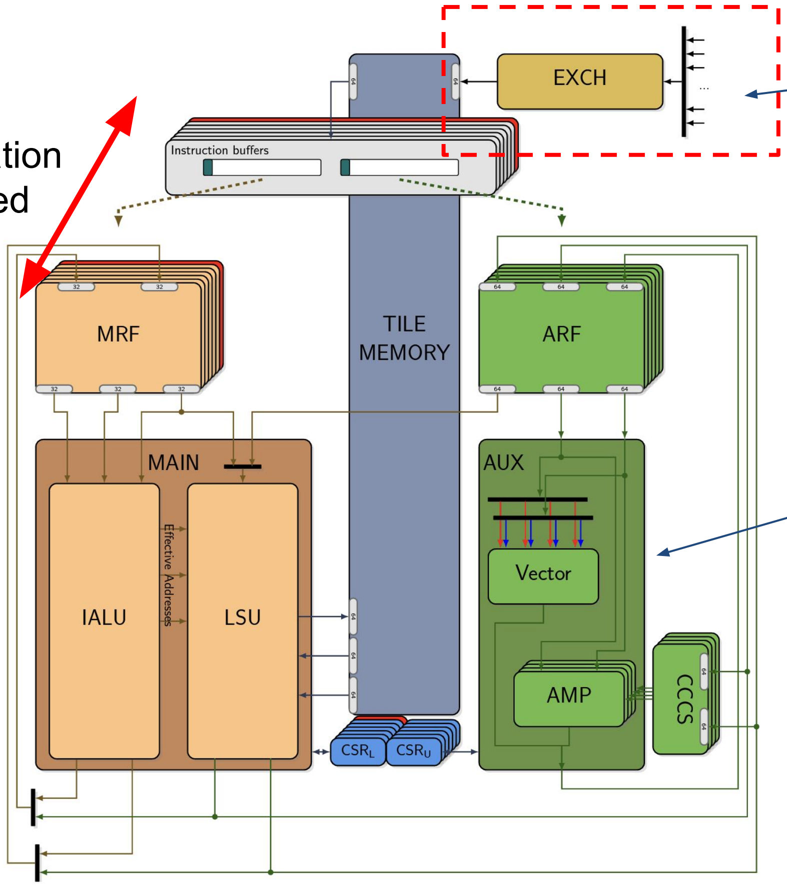
Integer operations, memory operations, control flow

[†] Very Long Instruction Word

Source: Tile Vertex ISA 1.2.3

The ISA uses VLIW [†] for the MAIN, and AUX pipeline

6 threads:
- ❑ **no** synchronization
- ❑ Time-multiplexed



No explicitly exposed API

Floating point unit, special AXPY instructions.

Very AI/ML workload centric.

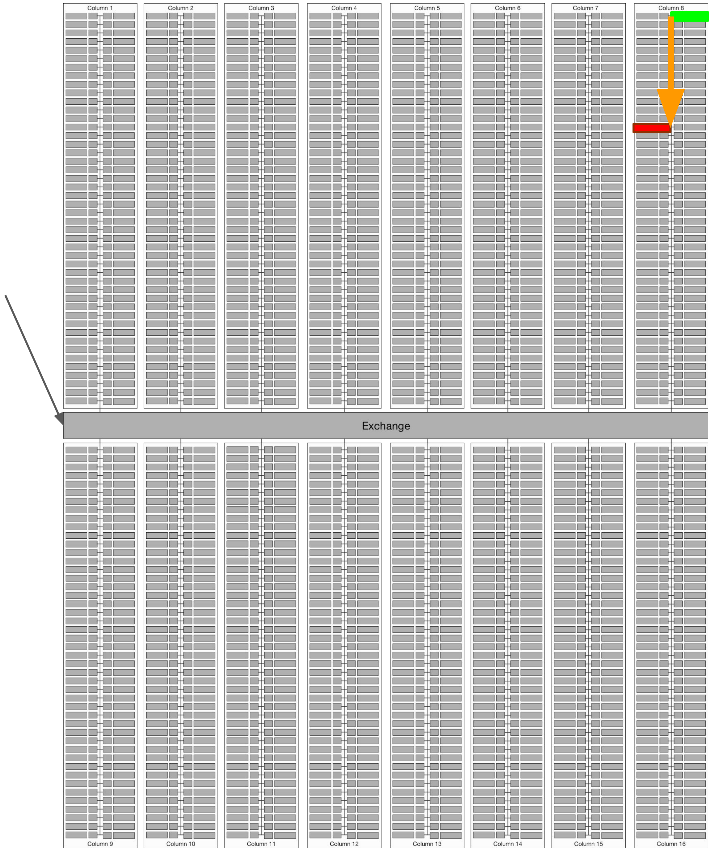
Integer operations, memory operations, control flow

[†] Very Long Instruction Word

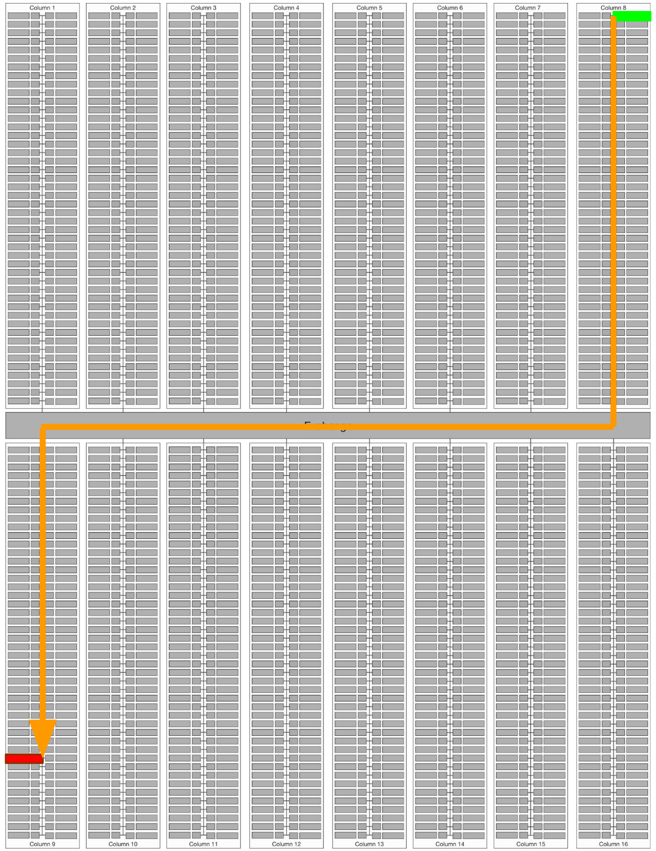
Source: Tile Vertex ISA 1.2.3

A 1:1 communication is possible

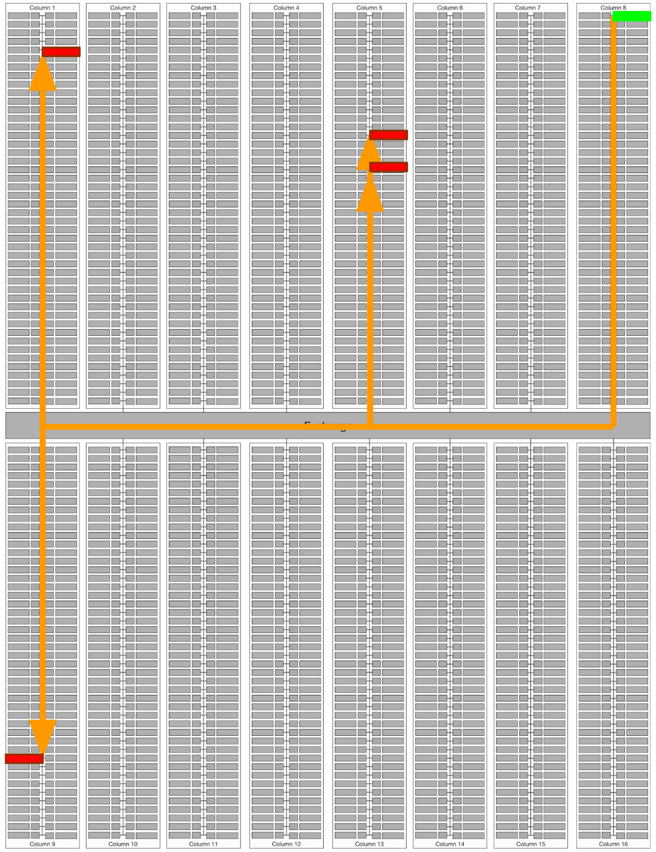
Crossbar Switch:
Tile-to-tile is *"constant"* latency



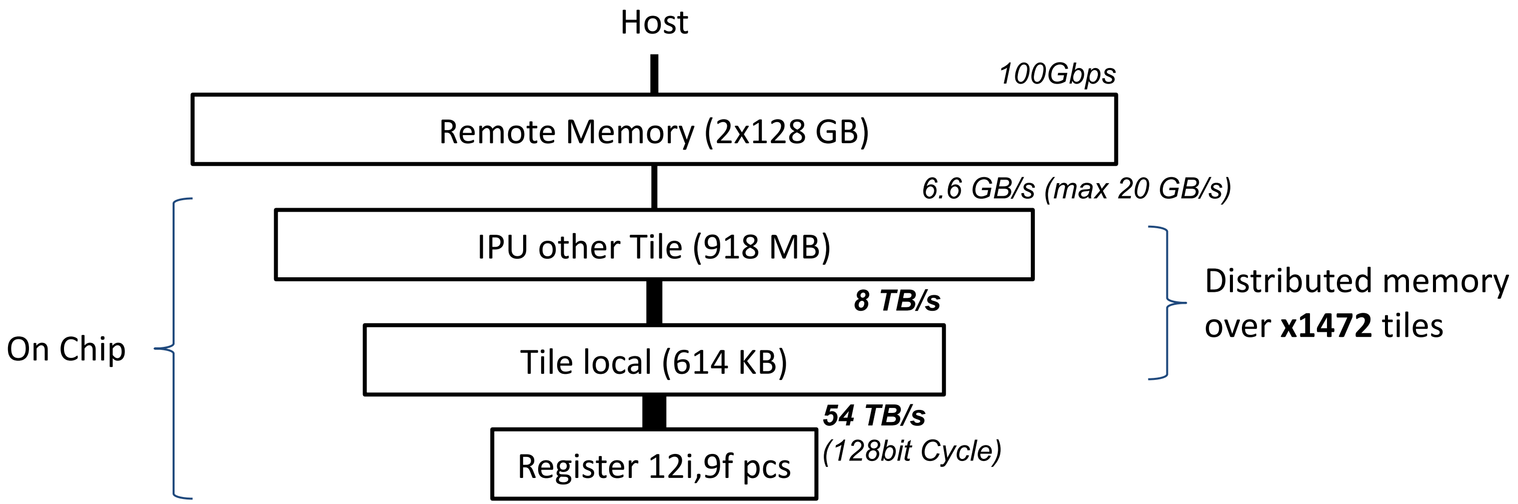
There is no restriction on the destination location



More complex communication patterns with broadcasts are possible



We have good throughput/latency only on the chip



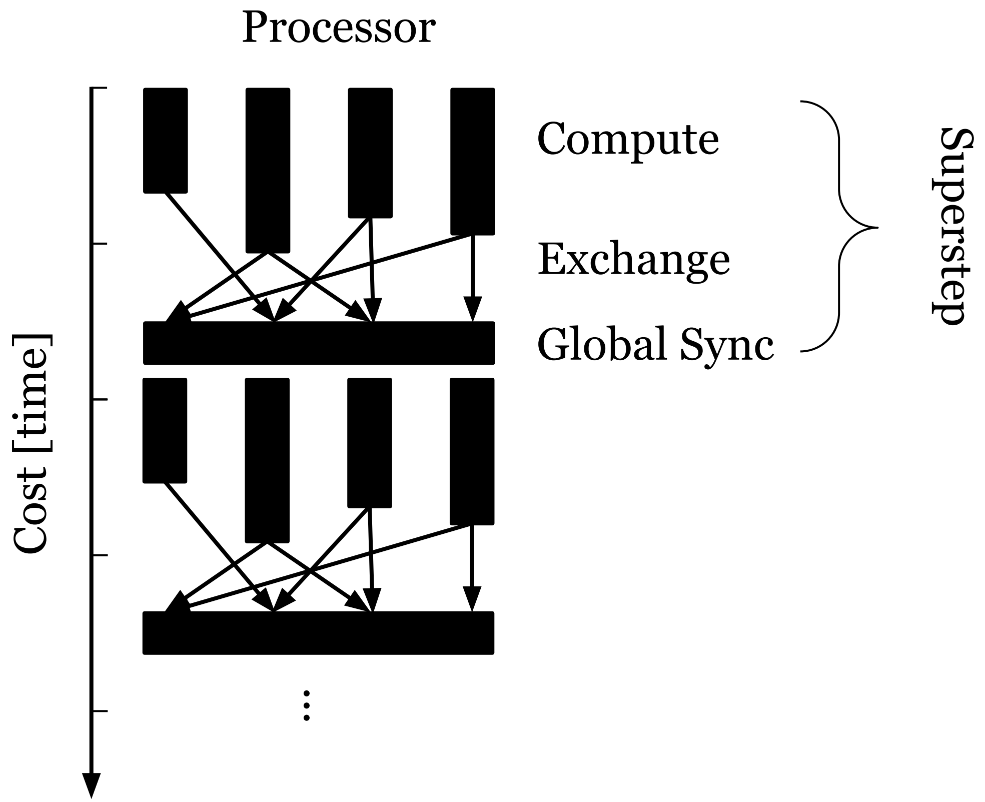
The Bulk-Synchronous Parallel (BSP) model is built into the hardware

Theory:

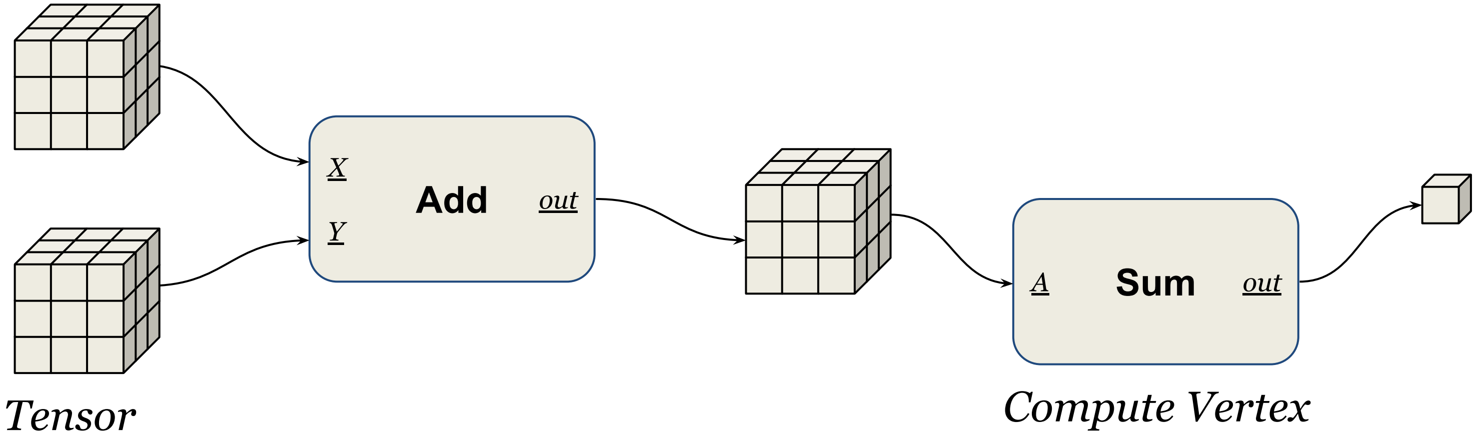
- Simple synchronization and coordination
- 3 Phases
  - Exchange
  - Compute
  - Sync

Applied:

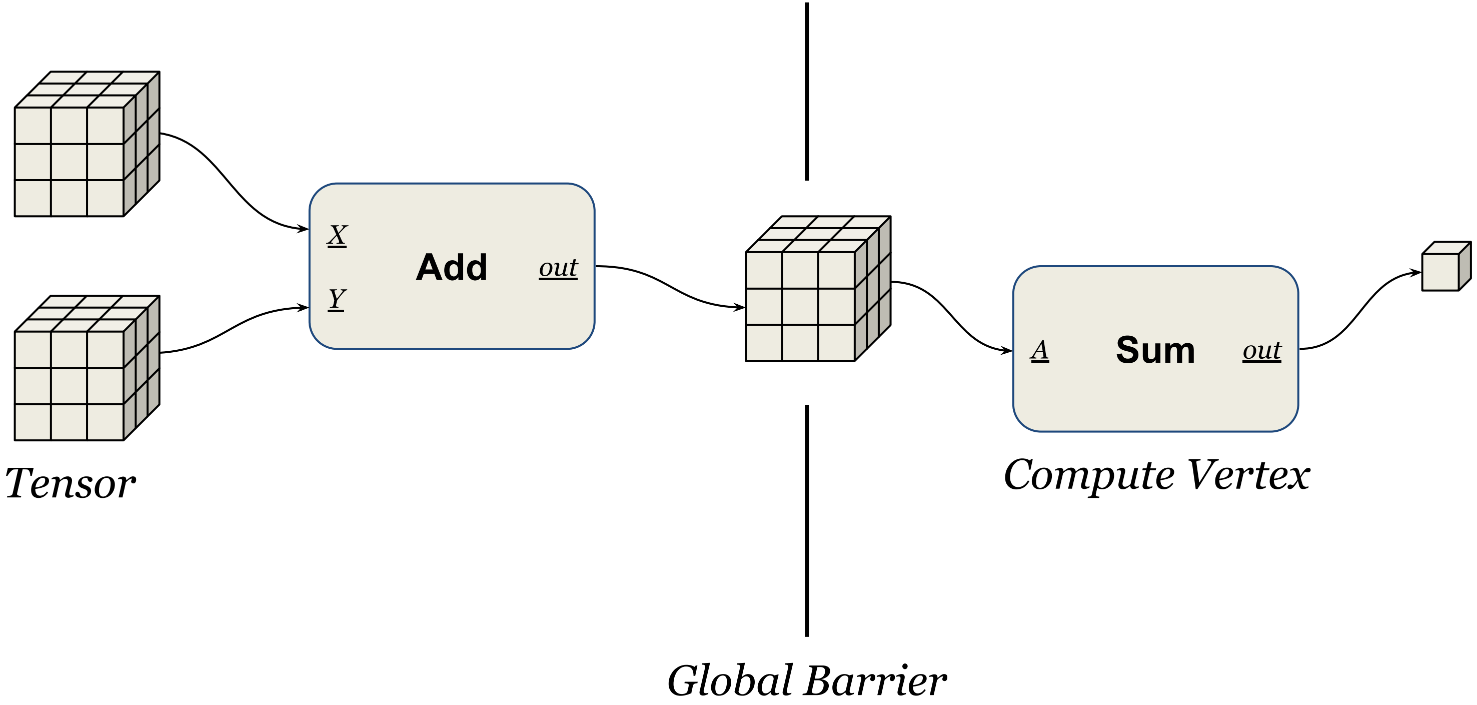
- Only pre-defined communication



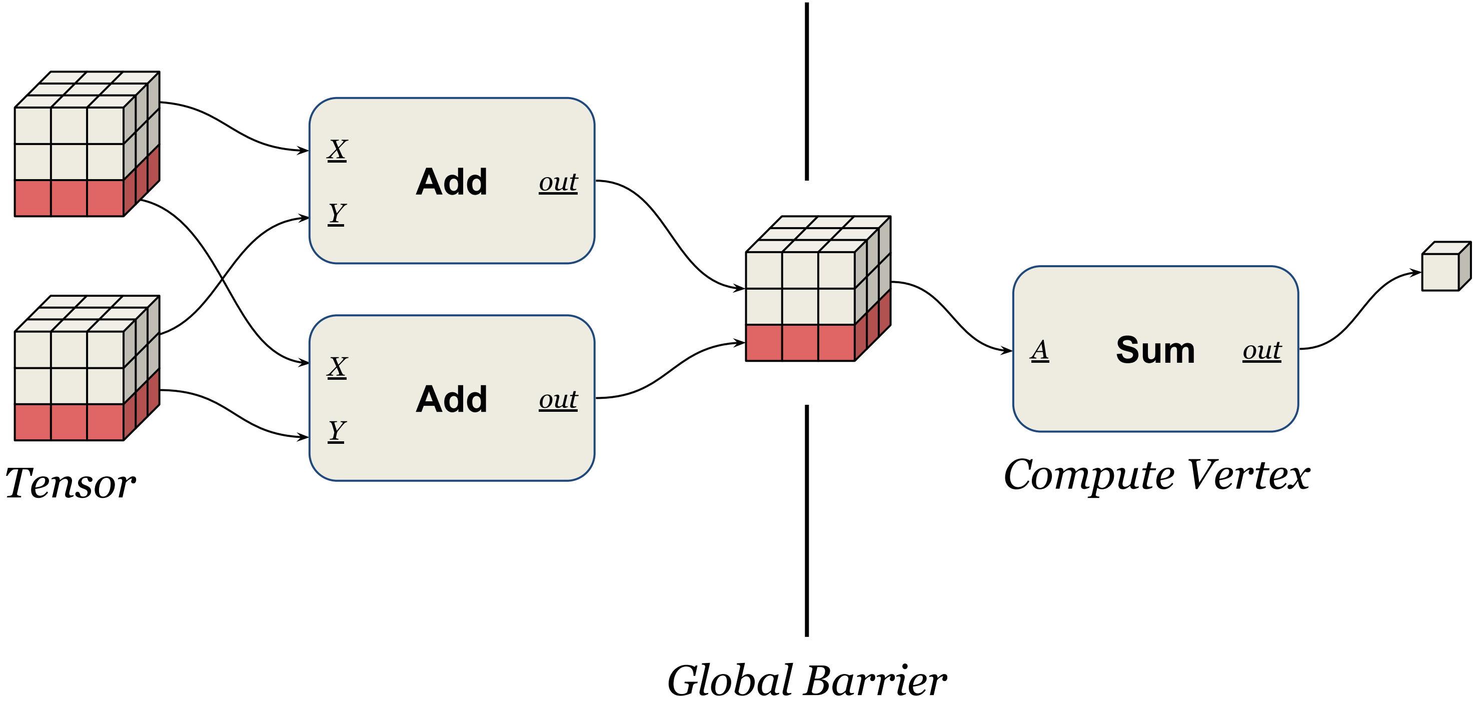
The computational graph indirectly defines exchanges from Tensor source location to Vertex input.



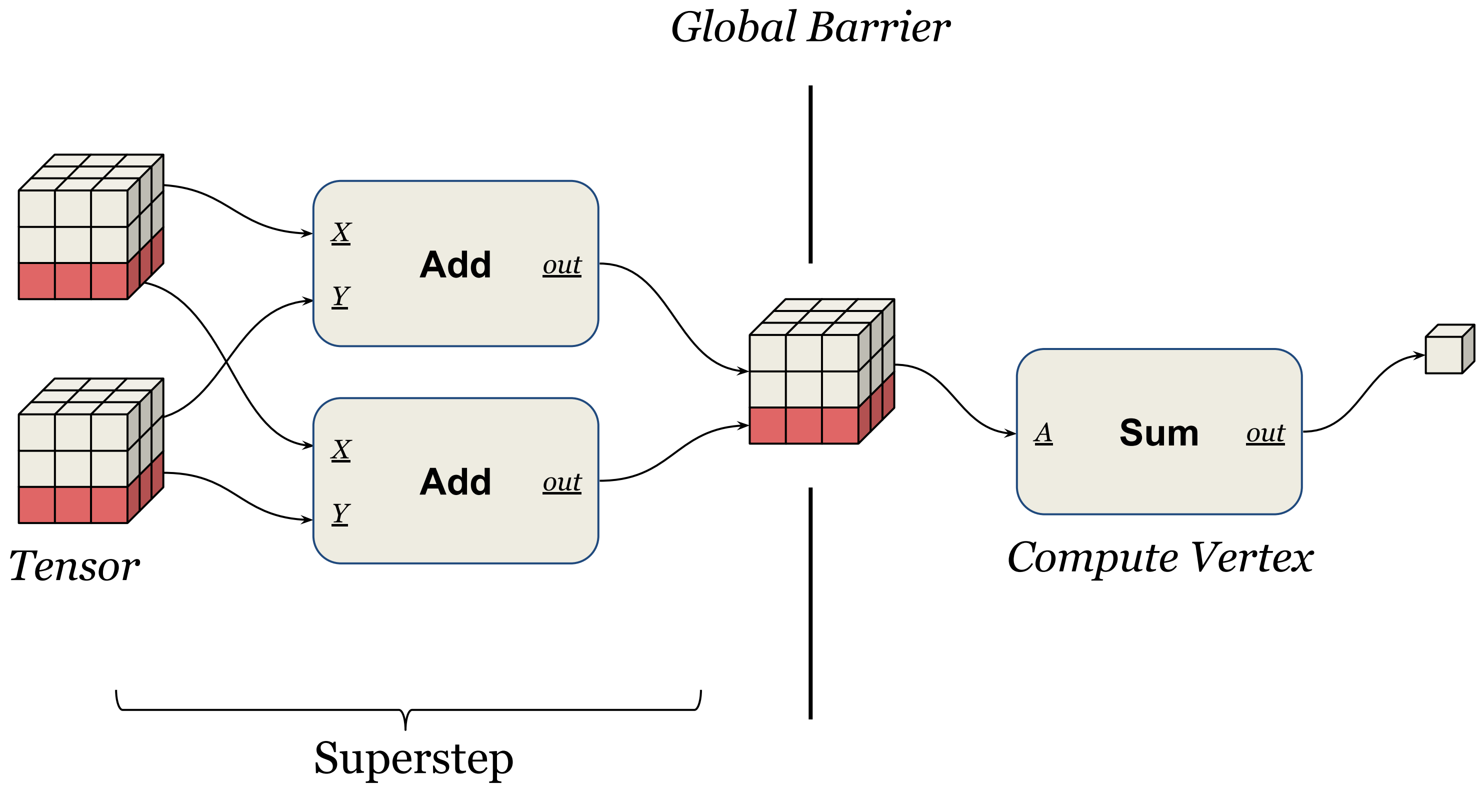
The computational graph indirectly defines exchanges from Tensor source location to Vertex input.



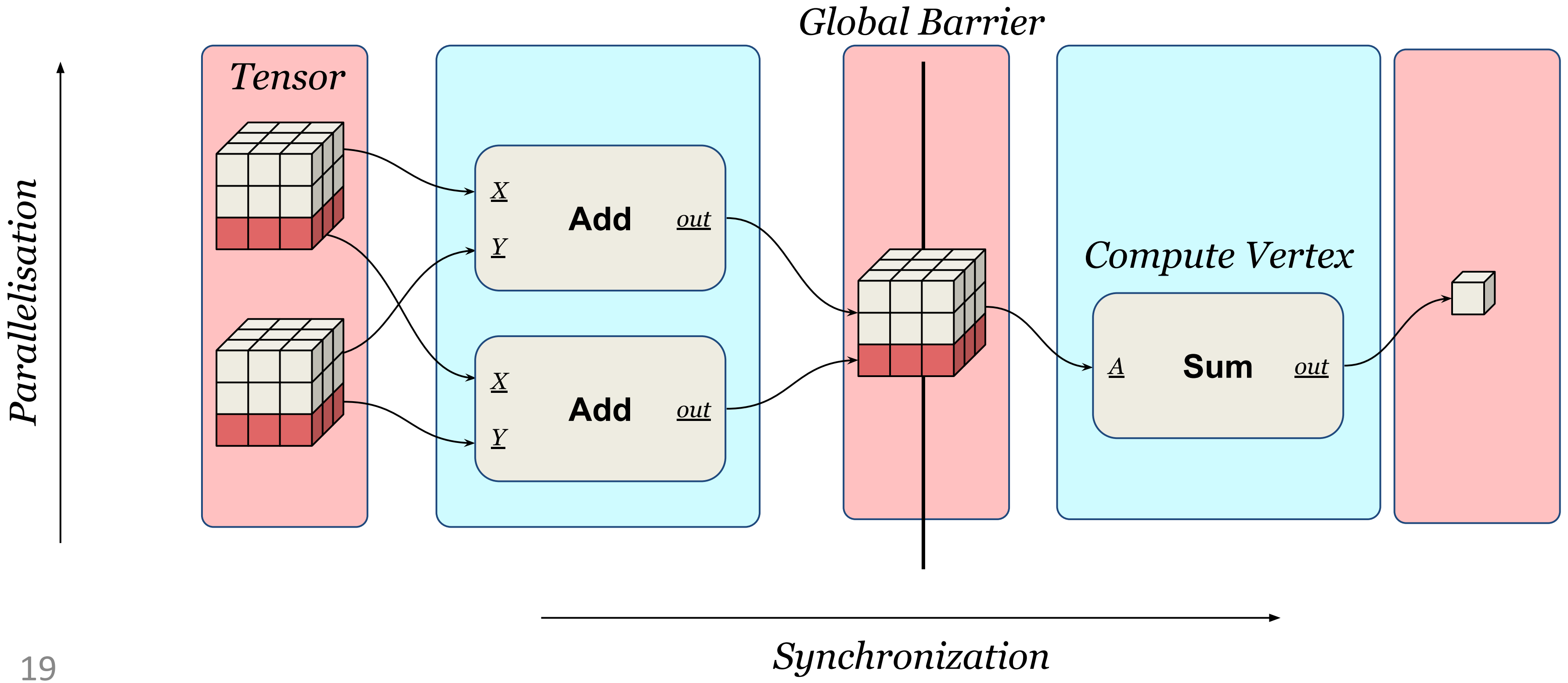
The computational graph indirectly defines exchanges from Tensor source location to Vertex input.



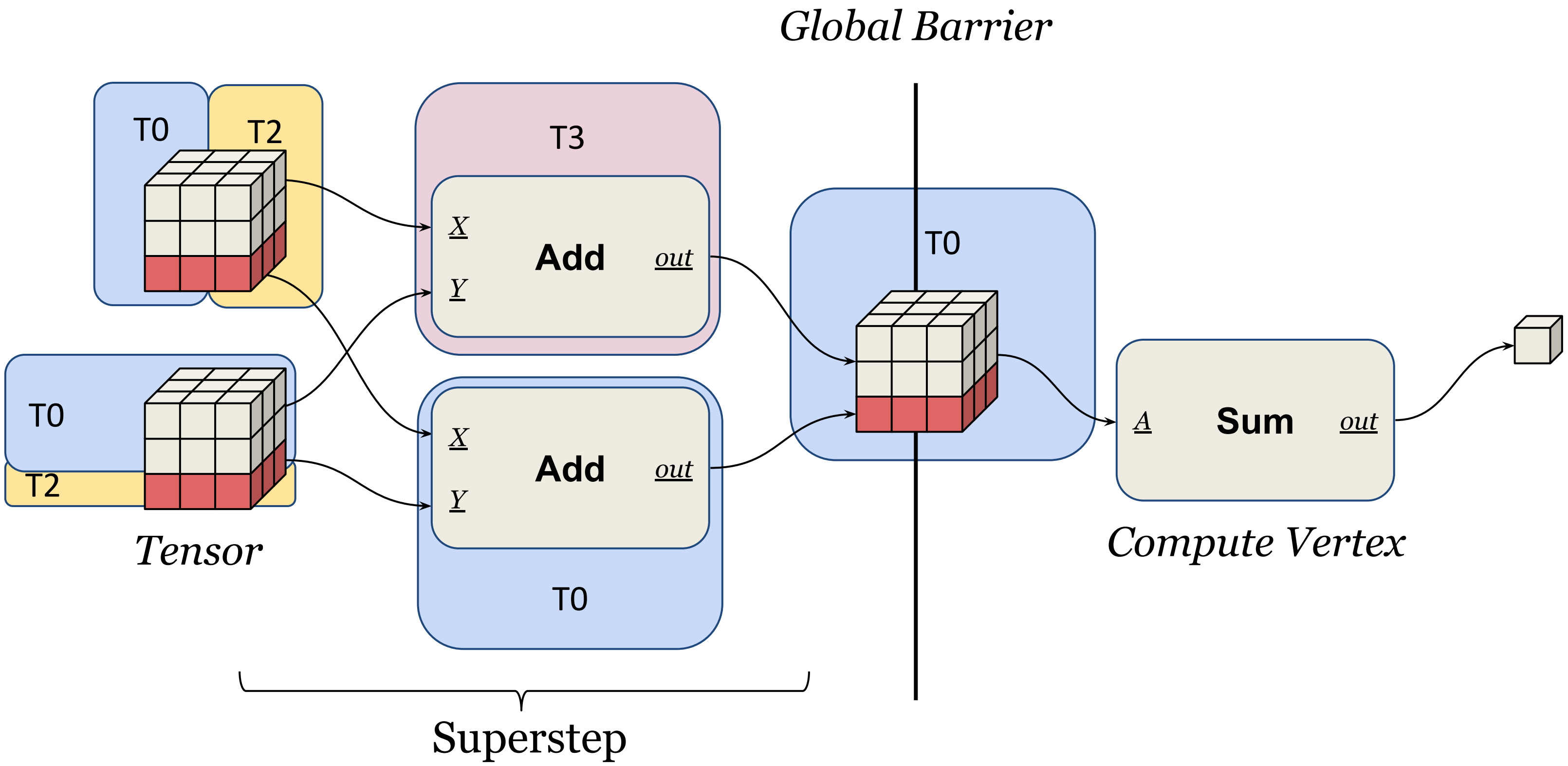
The computational graph indirectly defines exchanges from Tensor source location to Vertex input.



The IPU uses a dataflow model to define its computation and communication



Mapping has to be specified explicitly, the compiler creates exchange code



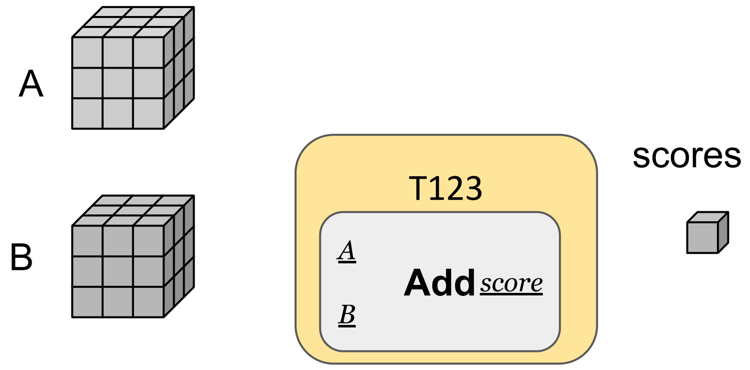
Tensors get copied to the tile running the codelet

```
// Compute graph types.
Tensor A{};
Tensor B{};
Tensor scores{};
```

```
// Add the codelet to a vertex.
VertexRef vtx = graph.addVertex(group, "Add");
graph.setTileMapping(vtx, 123);

// Connect the tensors.
graph.connect(vtx["A"], A);
graph.connect(vtx["B"], B);
graph.connect(vtx["score"], scores[0]);
```

Tensors get copied to the tile running the codelet

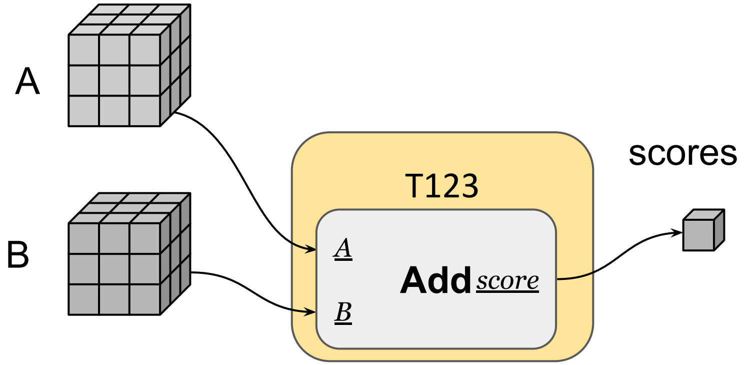
```
// Compute graph types.
Tensor A{};
Tensor B{};
Tensor scores{};

// Add the codelet to a vertex.
VertexRef vtx = graph.addVertex(group, "Add");
graph.setTileMapping(vtx, 123);

// Connect the tensors.
graph.connect(vtx["A"], A);
graph.connect(vtx["B"], B);
graph.connect(vtx["score"], scores[0]);
```



Tensors get copied to the tile running the codelet

Host Compile Tile

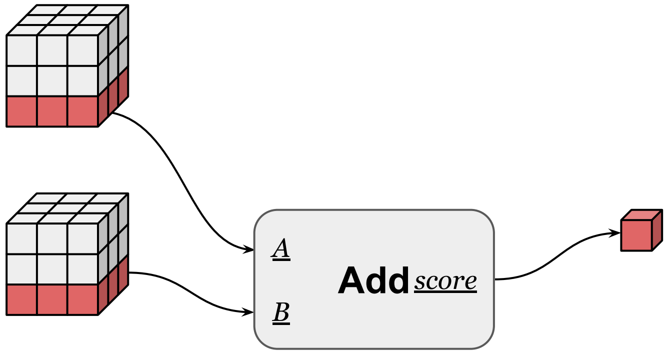
```
// Compute graph types.
Tensor A{};
Tensor B{};
Tensor scores{};

// Add the codelet to a vertex.
VertexRef vtx = graph.addVertex(group, "Add");
graph.setTileMapping(vtx, 123);

// Connect the tensors.
graph.connect(vtx["A"], A);
graph.connect(vtx["B"], B);
graph.connect(vtx["score"], scores[0]);
```



Codeletes are as C++ classes with a default entry function

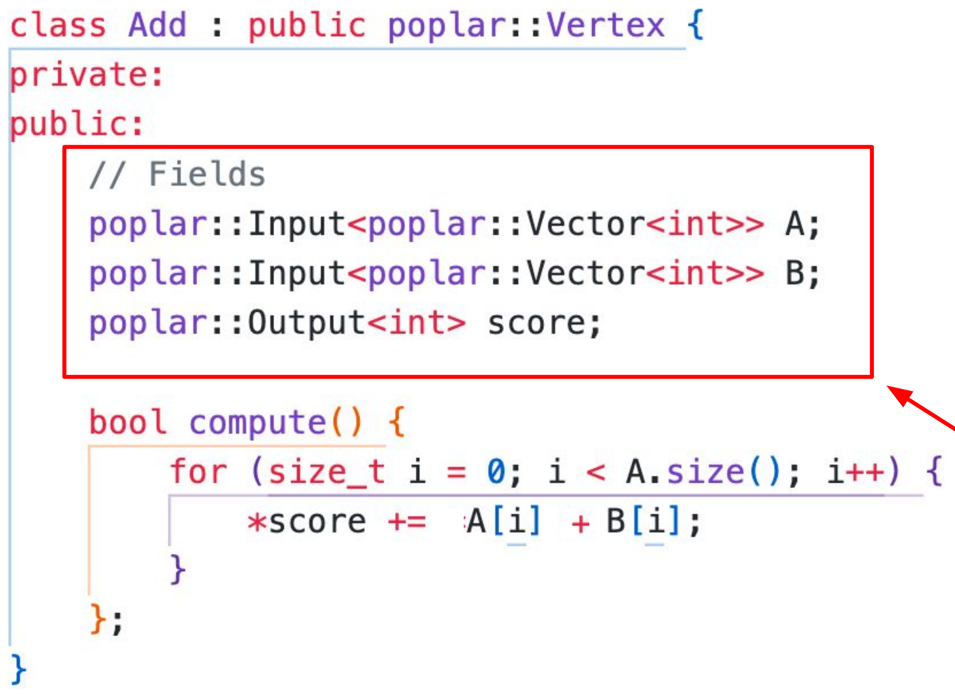
IPU Kernel Code

```
class Add : public poplar::Vertex {
private:
public:
    // Fields
    poplar::Input<poplar::Vector<int>> A;
    poplar::Input<poplar::Vector<int>> B;
    poplar::Output<int> score;

    bool compute() {
        for (size_t i = 0; i < A.size(); i++) {
            *score +=  A[i]  + B[i];
        }
    };
}
```

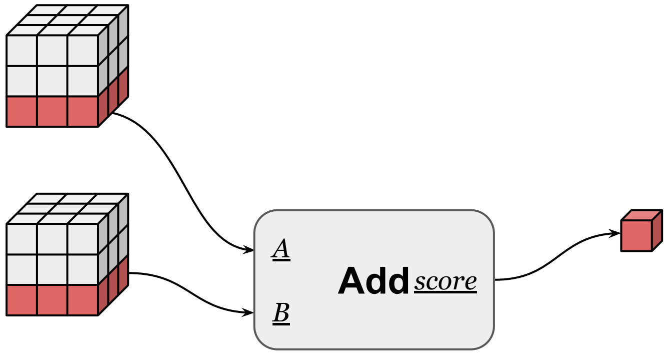


Codelets are as C++ classes with a default entry function

IPU Kernel Code



```
class Add : public poplar::Vertex {
private:
public:
    // Fields
    poplar::Input<poplar::Vector<int>> A;
    poplar::Input<poplar::Vector<int>> B;
    poplar::Output<int> score;

    bool compute() {
        for (size_t i = 0; i < A.size(); i++) {
            *score +=  A[i]  + B[i];
        }
    };
}
```



The compiler generates code to exchange these members defined by the *tile mappings* in the dataflow graph

Much research has been done on the topic of sequence alignment

BLAST

FASTA

minimap2

burrows wheeler alignment

Heuristics

Alpern 1995

Farrar 2007

SSW Library, 2013

SIMD Implementations

cudaSW++

GASAL2

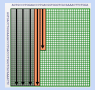
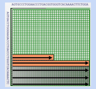
ADEPT-SW

GPU

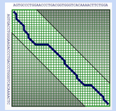
FGPA

Cell BE

Dedicated hardware

$O(n)$ space



$O(n^2c)$ time

Tradeoffs

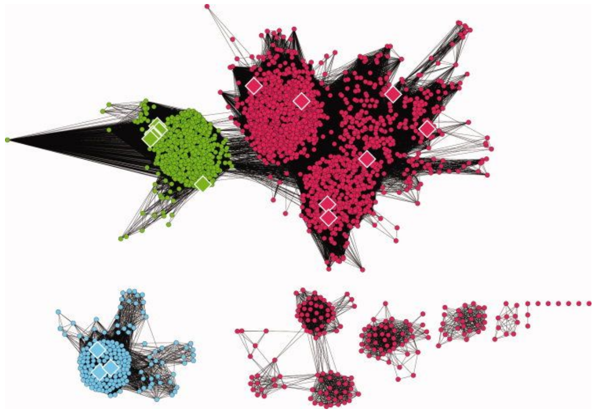
Smith Waterman Implementations

…

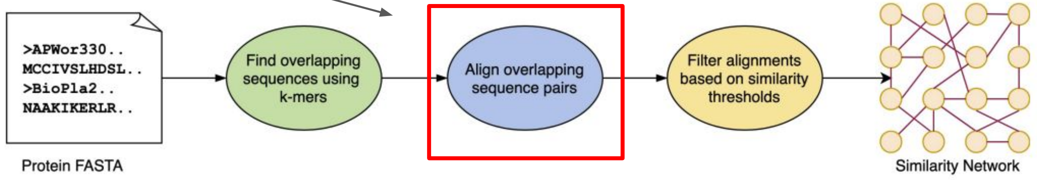
Needleman-Wunsch Implementations

Exact Algorithms

PASTIS a real-world protein clustering pipeline application



Many-to-many
Smith-Watermann
Sequence Alignment



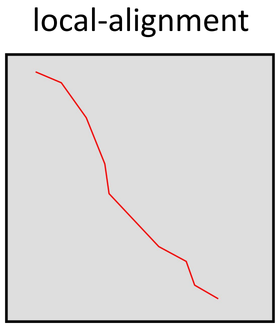
>APWor330..
MCCIVSLHDSL..
>BioPla2..
NAAKIKERLR..
Protein FASTA
Find overlapping sequences using k-mers
Align overlapping sequence pairs
Filter alignments based on similarity thresholds
Similarity Network

PASTIS pipeline

Source:
Selvitopi, Oguz, et al. "Distributed many-to-many protein sequence alignment using sparse matrices." *SC20,* IEEE, 2020.
Selvitopi, Oguz, et al. "Extreme-scale many-against-many protein similarity search." *SC22*, IEEE, 2022.

The Smith-Waterman algorithm

- ❑ Local Alignment Algorithm to find **the best** matching **overlap**
- ❑ No fixed start/end position
  - ❑ This is different to the Needleman-Wunsch algorithm
- ❑ Affine gap penalties make is difficult to compute
  - ❑ (i.e. a longer gap is more likely than many conjunct gaps)
- ❑ Proteins benefit from similarity scoring, valuing indels per basis
  - ❑ i.e. BLOSUM62



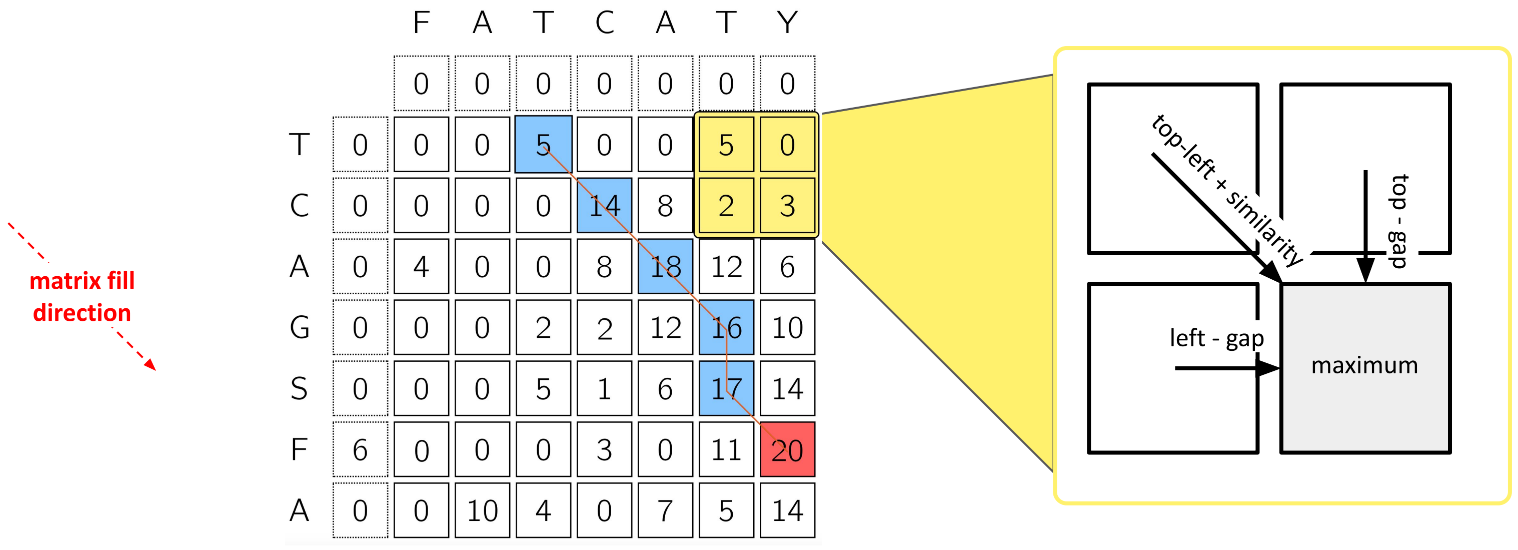
```
FATCA-TY
  ||| ||
  TCAGSFA
```

We include symbol similarity

→ Smith-Waterman *based* algorithms with affine gaps and similarity matrices offer good quality for protein sequences but are slow

The Smith-Waterman algorithm

- Dynamic Programming Algorithm
  - We create a matrix containing scores
- The highest score indicates the best valued alignment of two sequences
- Cell updates need the top, top-diagonal, and left fields value

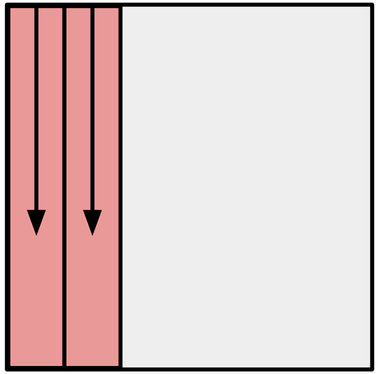


		F	A	T	C	A	T	Y
		0	0	0	0	0	0	0
T	0	0	0	5	0	0	5	0
C	0	0	0	0	14	8	2	3
A	0	4	0	0	8	18	12	6
G	0	0	0	2	2	12	16	10
S	0	0	0	5	1	6	17	14
F	6	0	0	0	3	0	11	20
A	0	0	10	4	0	7	5	14

Smith-Waterman implementation for the IPU

We choose a $\boldsymbol{O(n)}$ memory formulation for our implementation

- ❑ Only columns need to be stored
- ❑ No on tile SIMD → Wavefront algorithm is not helpful



Careful coding and type (INT/FP) utilization to use the VLIW

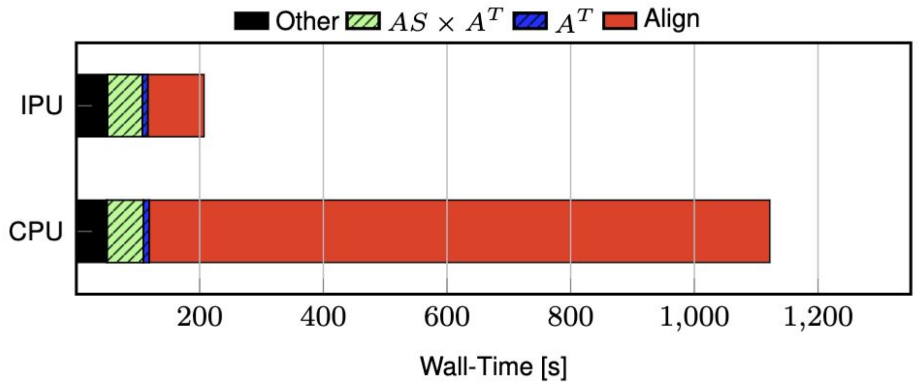
Single sequence comparison per thread

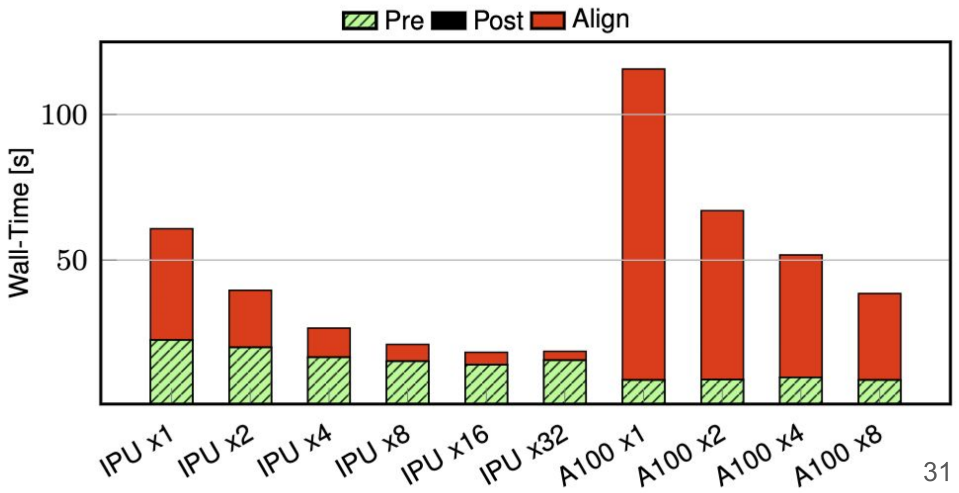
→ No communication as whole comparison fits in SRAM domain (tile memory)

Balance $|\boldsymbol{A}|*|\boldsymbol{B}|$ complexity due to BSP-makespan limitation

PASTIS results

- 5x speedup vs CPU for total pipeline
  - CPU: 1142s, 88% alignment time
  - IPU: 225, 40% alignment time
    - Alignment speedup of 11.1x
- 24.9x speedup vs GPU in kernel
  - 2.8x 1IPU/1GPU
  - 24.9x 16IPU/1GPU
  - 6.9x 16IPU/8GPU
- Our Alignment scales linearly with number of IPUs up to 16 devices

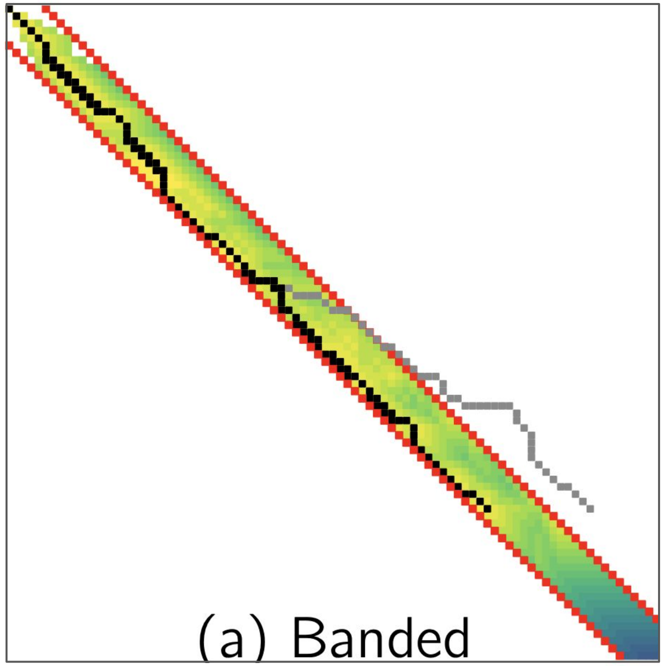


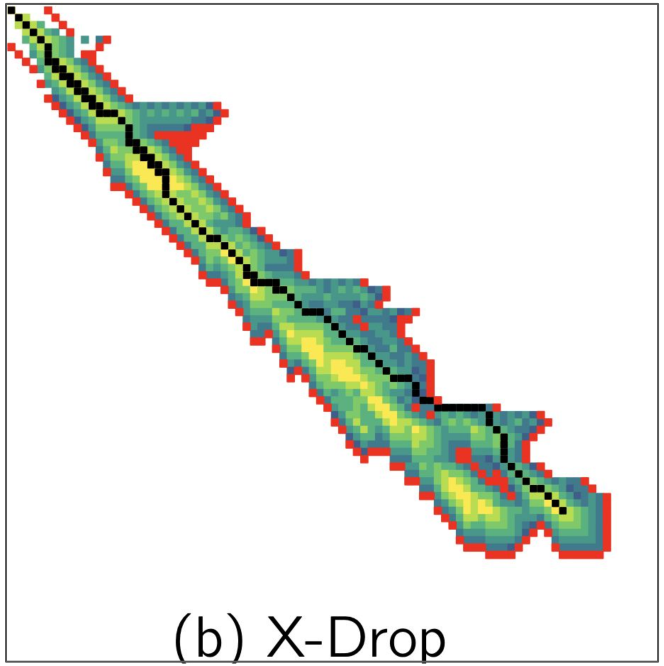


Final work is under review and includes further details on

- ❑ Detailed discussion of the algorithm
  - ❑ Bespoke IPU implementation
- ❑ PASTIS and MetaHipMer2 pipeline showcase
- ❑ Single device comparison to CPUs and GPUs
  - ❑ 2 GPU implementation
  - ❑ 3 CPU implementations
- ❑ Strong&Weak scaling results
- ❑ Discussion on load-balancing algorithms

Seed extesion and X-Drop reduces the area compared to SW





Heuristic optimization to reduce the search area, makes it difficult for GPUs, wide SIMD

(a) Static search area reduction

(b) X-Drop dynamically reduces the search for “unrecoverable” bad values

X-Drop Insights

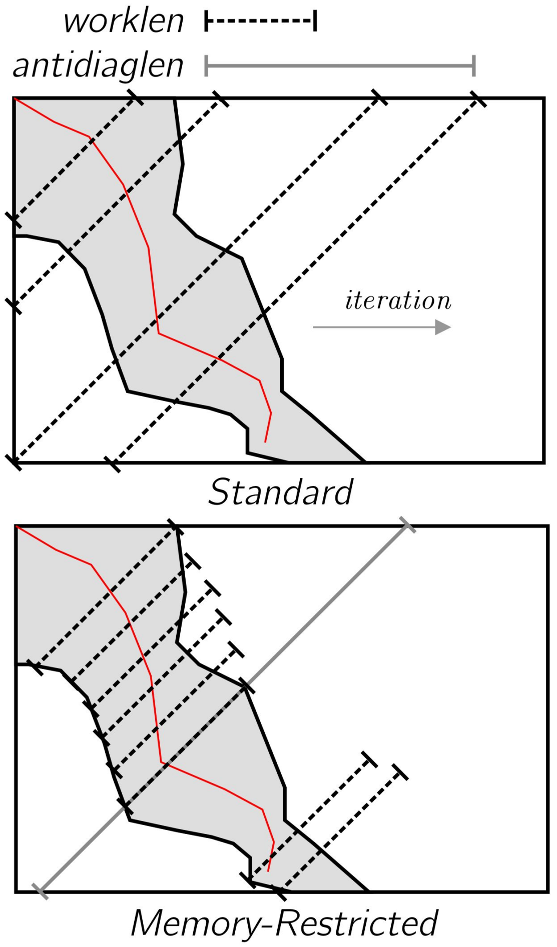
- Tailored to longer sequences 10k-20k symbols+
  - Worse SIMD suitable than simple Smith-Waterman/Needleman-Wunsch
- Terminates fast on mismatching sequences
- Higher sequence error rate/similarity $\rightarrow$ larger searchable area size
- Memory requirements of normal X-Drop implementations are $\boldsymbol{O(N)}$
  - More specifically $\boldsymbol{3*N}$
  - Challenge: Out of memory for 6 thread requiring algorithm scratch space

X-Drop Observation, only a small part of the temporary workspace is needed

- The active worklen is only written each phase and read next phase (grey area)

We can reformulate X-Drop to only allocate the maximum worklen and work with reduced memory

→ **55x reduction** in memory



Optimizing the memory usage allows us to place more problems to a single tile and utilize parallelism

- ❑ Due to early termination balancing becomes challenging
  - ❑ Increase inputs (samples) to reduce variance

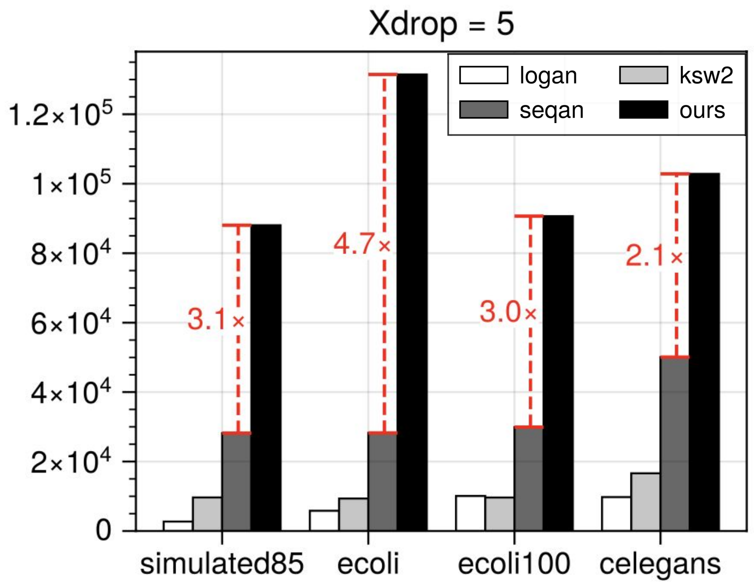
For X values from 5 to 50

- ❑ 1.7x to 4.7x against state-of-the-art CPUs and codes (Milan 7763 64 Threads)
- ❑ 7x to 22x against only GPU code (A100)

Real world pipelines (alignmen kernels):

ELBA: 22.3x 16 IPUs vs 16 GPUs (C elegans HiFi)

PASTIS: 4.7x speedup (metaclust 500k)



Final work is under review and includes further details on

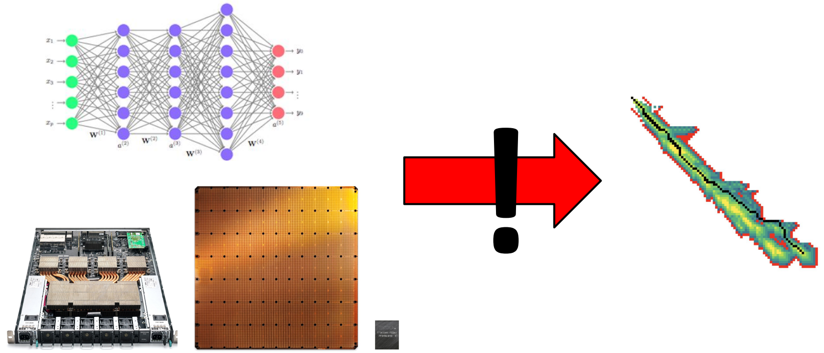
- ❑ Detailed discussion of the algorithm
- ❑ Analysis of the memory efficiency under dataset and X parameters
- ❑ ELBA and PASTIS pipeline showcase
- ❑ Single device comparison to CPUs and GPUs
  - ❑ 3 CPU implementations
  - ❑ 1 GPU implementations
  - ❑ 2 IPU generations
- ❑ Strong&Weak scaling results
- ❑ Many-to-Many sequence reuse for further memory reduction
  - ❑ 3-4x transfer savings

Reusing AI/ML-Accelerators for Sequences alignment problems is possible and beneficial

Sequence alignment algorithms are fundamentally memory bound and require many instructions

$\downarrow$

SRAM-based processing offered by AI accelerators offer memory and instruction throughput, but require careful memory management



Questions?