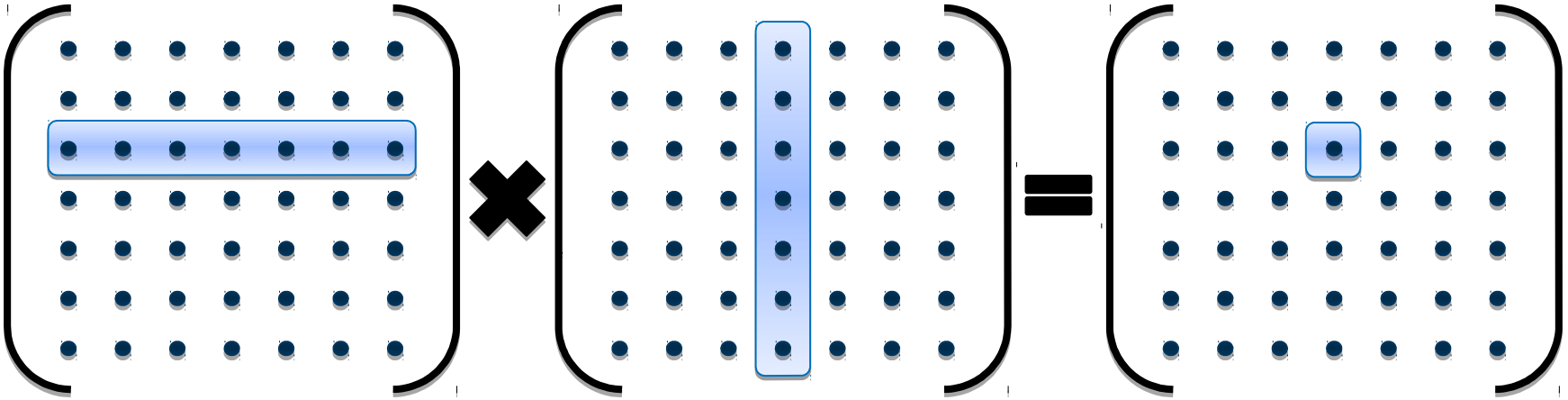
Dense Matrix Multiplication

Feb 2015

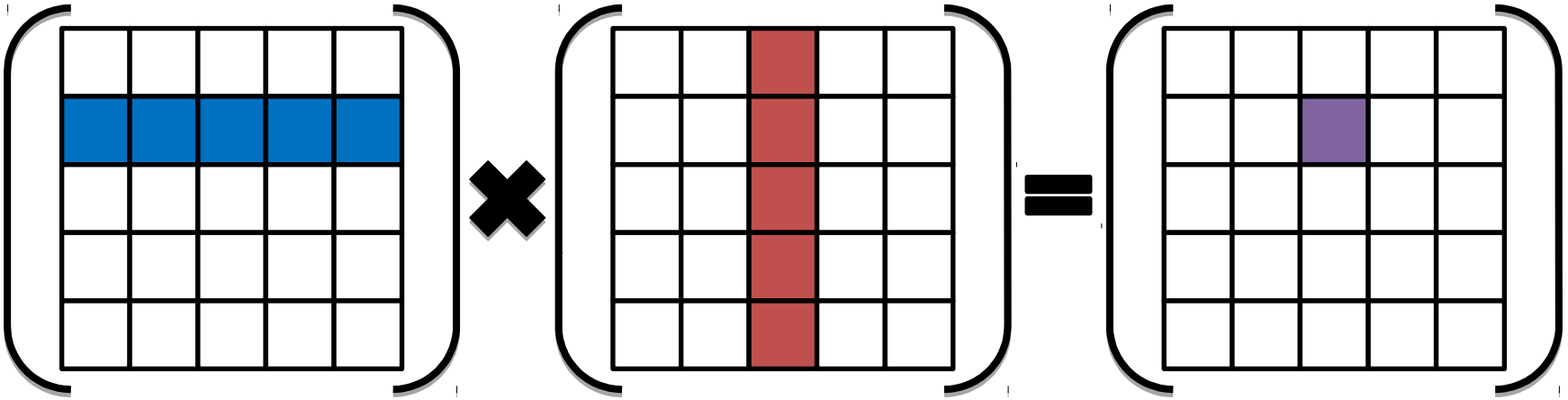
Overview

- A matrix is “dense” if most of its elements are non-zero.
- Multiplication involves a dot product between every row of a matrix against every column of another matrix
- This leads to $O(N^3)$ compute with only $O(N^2)$ data
- Since compute costs dominate over data transfer costs, this is ideal for DFE acceleration



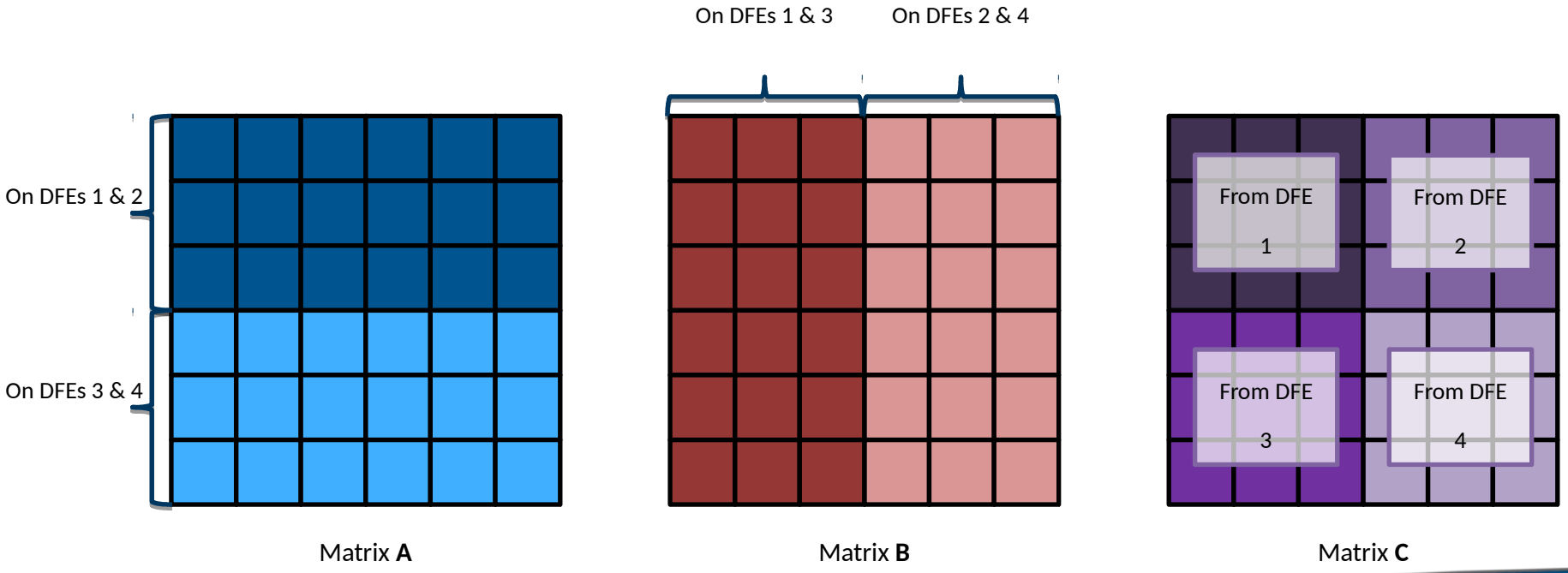
Tiling

- A naive implementation of matrix multiplication takes each row and each column and computes a dot product
- This leads to having 2 inputs for each multiply-accumulate, which would be memory bound
- It is possible to decompose the matrices into tiles to increase data reuse so that we are compute bound
- An NxN tile from each matrix is used to produce all N^2 dot products, so we perform $2N^3$ FLOPs on $2N^2$ data to produce N^2 partial results



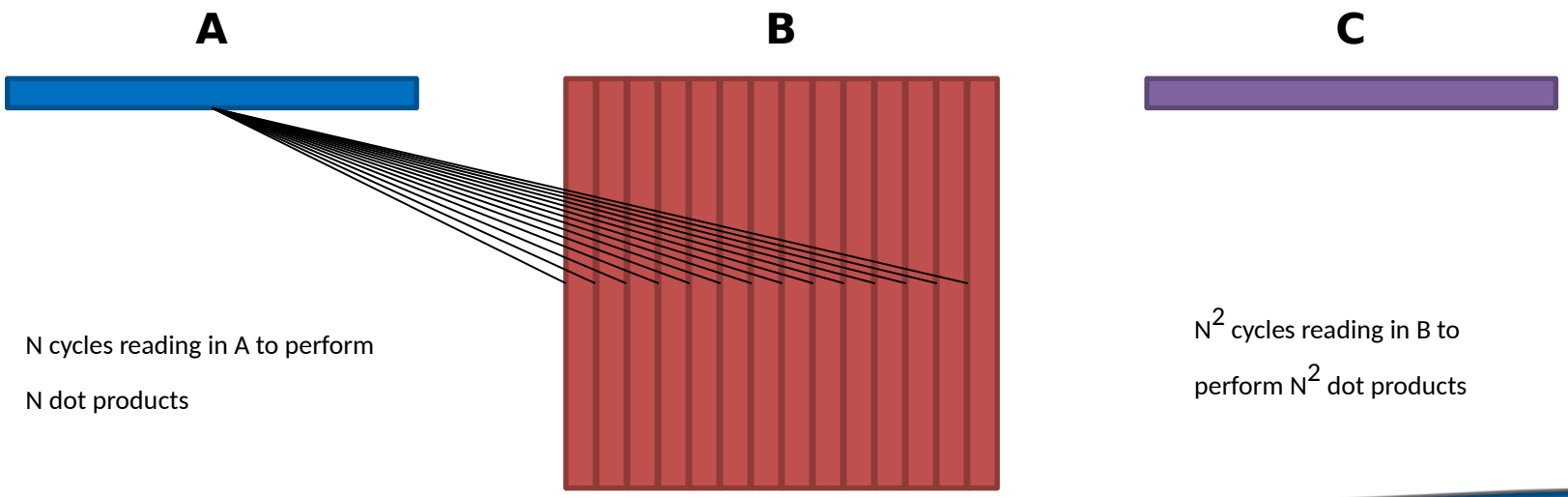
Tiling Between DFEs

- To produce a tile of C we need a whole row of tiles from A and a while column of tiles from B
- A single row of tiles from A interacts with every column of B
- When decomposing across multiple DFEs, each DFE will produce a section of C, but sections of A and B must be duplicated across DFEs
- Below is an example for 4 DFEs:



Tile Buffers

- A tile from matrix B is loaded into FMem on the DFE, and a single row of a tile of A is loaded into registers
- Next all N row-column dot products are computed, with one dot product (of size N) performed on every cycle
- For tile size N^2, N^2 dot products are performed, so it only necessary to read in one element from each matrix per cycle
- Tiles from B are double-buffered so that tiles can be loaded in parallel with computation



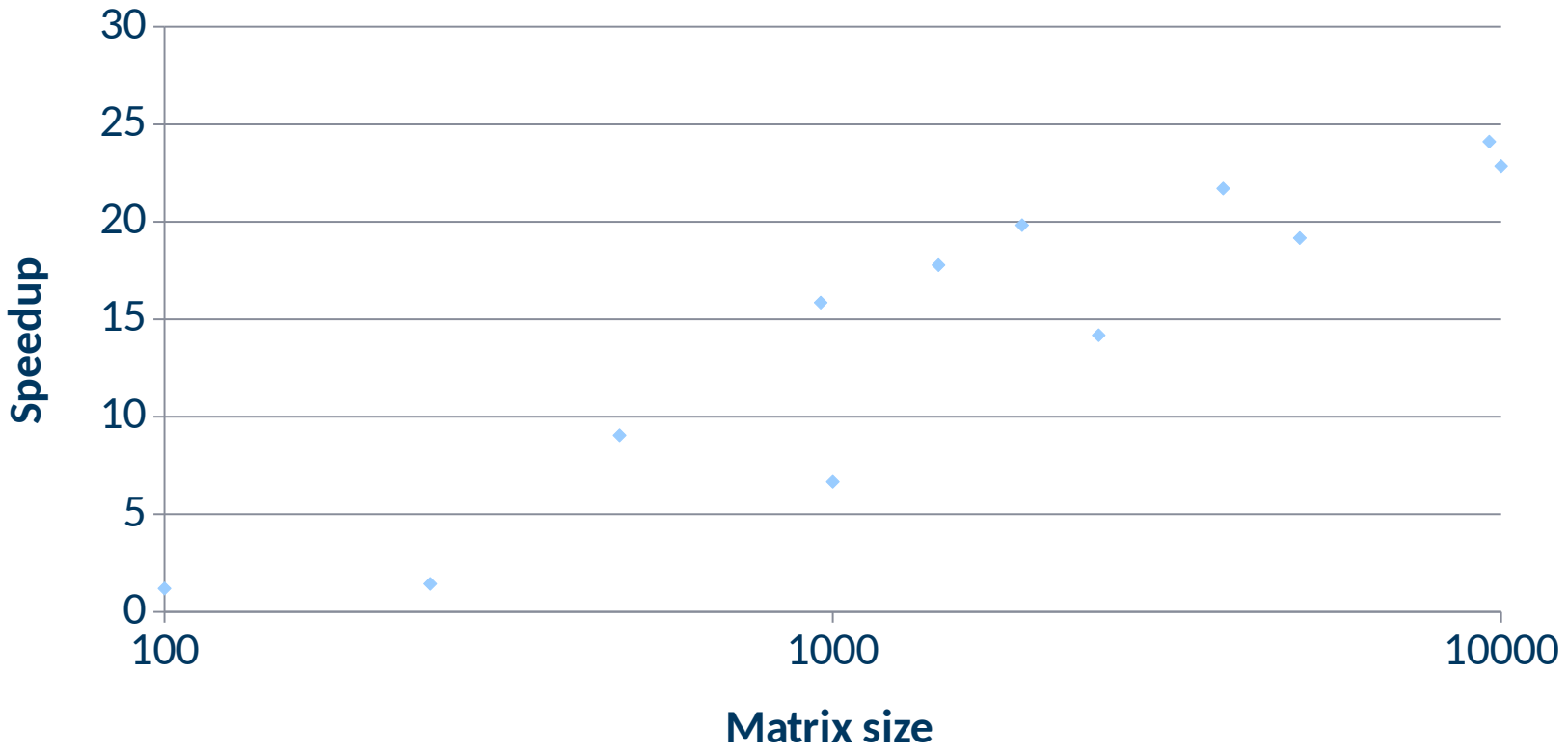
Tile Size

- To achieve best performance, we wish to perform as many parallel computations as possible
- The number of parallel computations is equal to the tile size, so this equates to maximising tile size
- The possible limitations on block size are:
  - FMem for storing tiles of B
  - DSPs for computing multiplications
  - Logic (LUTs and FFs) for computing additions
- The design is also simpler if the block size fits into an exact number of LMem bursts
- MAX4 Maia has 1963 27x27 multipliers, which allows for 490 double precision multiplications per cycle
- A tile size of 480x480 was chosen for the following reasons:
  - This uses around 98% of the multipliers, 84% of Fmem and ~90% logic
  - It is exactly 9,600 LMem bursts

Tile Accumulation

- Accumulation of the partial results computed for C can be done in 3 ways:
  a) Use FMem store partial results (a whole tile of C)
  b) Use LMem to store intermediate results
  c) Perform accumulation on the CPU
- Option a) uses too much Fmem, so the tile size would need to be reduced
- Option b) is the best solution for many applications
- However, for simplicity, this app currently does c)

Speedup



Speedup DFE vs CPU:

- DFE: Single MAX4 MAIA DFE at 200MHz with 480 tile size
- CPU: ATLAS SSE3 BLAS on single core of Intel Xeon E5540

Possible Improvements

- Perform accumulation on DFE
  - Storing all 3 matrices in LMem eliminates PCIe and Infiniband bandwidth constraints
  - This allows for higher clock frequencies
  - Also decreases CPU load, allowing for more DFEs to be driven from a single CPU
- Use multiple kernels with smaller tiles
  - This would work more efficiently on small matrices
  - This would require more bandwidth into the chip, so requires matrices to come from LMem or a network connection
- Write software to optimally decompose matrices over any number of DFEs