CO405H

Computing in Space with OpenSPL

Topic 2: From temporal code to spatial graphs

Oskar Mencer **Georgi Gaydadjiev**

Department of Computing
Imperial College London

http://www.doc.ic.ac.uk/~oskar/
http://www.doc.ic.ac.uk/~georgig/

CO405H course page: http://cc.doc.ic.ac.uk/openspl14/
WebIDE: http://openspl.doc.ic.ac.uk
OpenSPL consortium page: http://www.openspl.org

o.mencer@imperial.ac.uk g.gaydadjiev@imperial.ac.uk

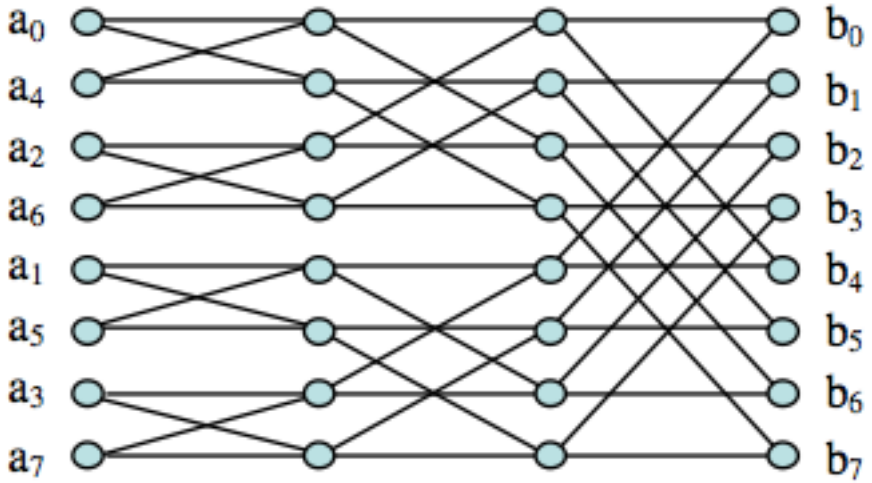
Overview

- Turn Expressions into Executable Graphs
- Flowing numbers through graphs
- Spatial memory system and data structures

Example: Fast Fourier Transform (FFT)

$$b_j = \sum_{k=0}^{n-1} a_k \cdot \omega_n^{kj} \quad for \quad j = 0,1,\ldots,n-1$$ Discrete Fourier Transform (DFT)

Many multiple-accumulate operations with time complexity of O(n^2)
FFT (Cooley-Tukey) is a less complex version (O(n log n)) that is

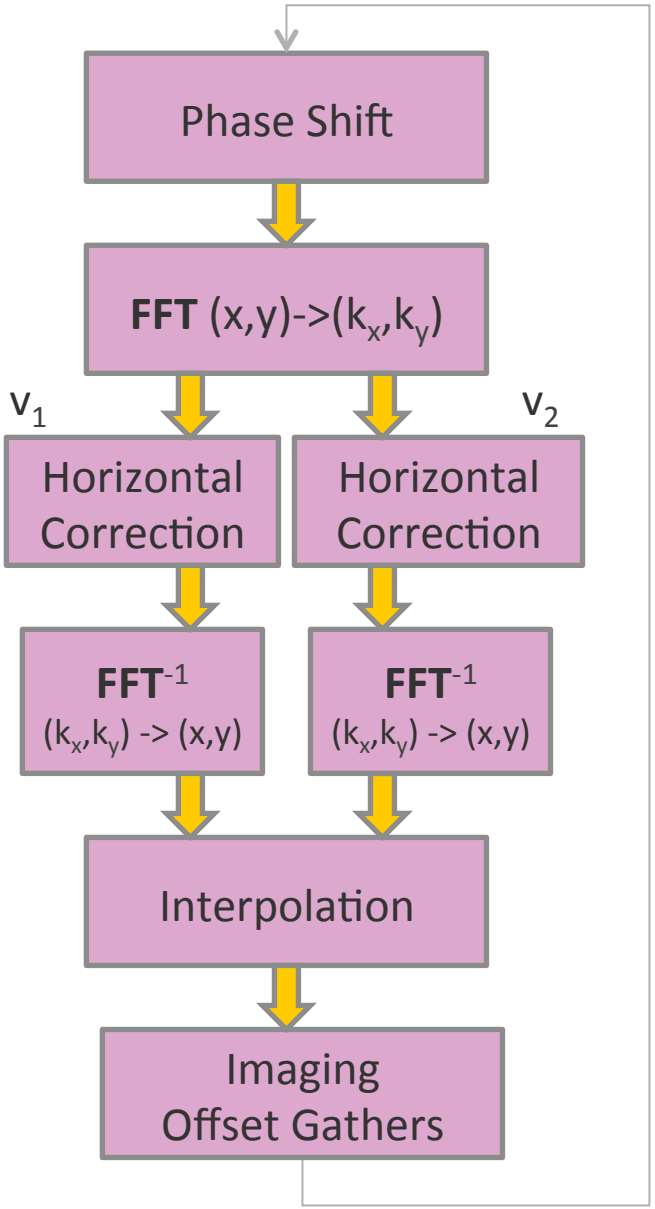


very suitable for 2D implementation in space as a set of interconnected MAC units

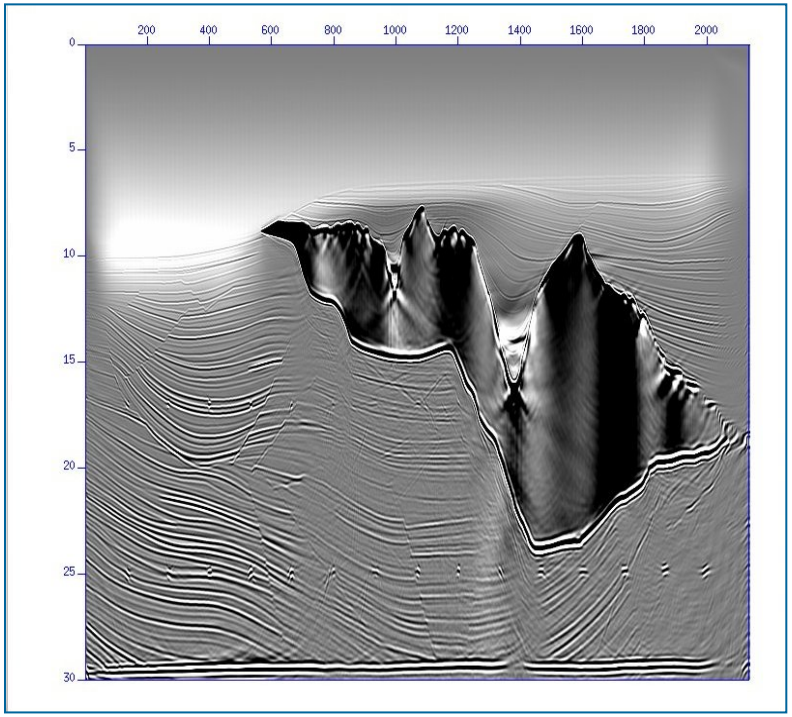
* MAC – Multiply Accumulate

C. Tomas et al, 2012

PSPI Algorithm



Loop over z depth layers



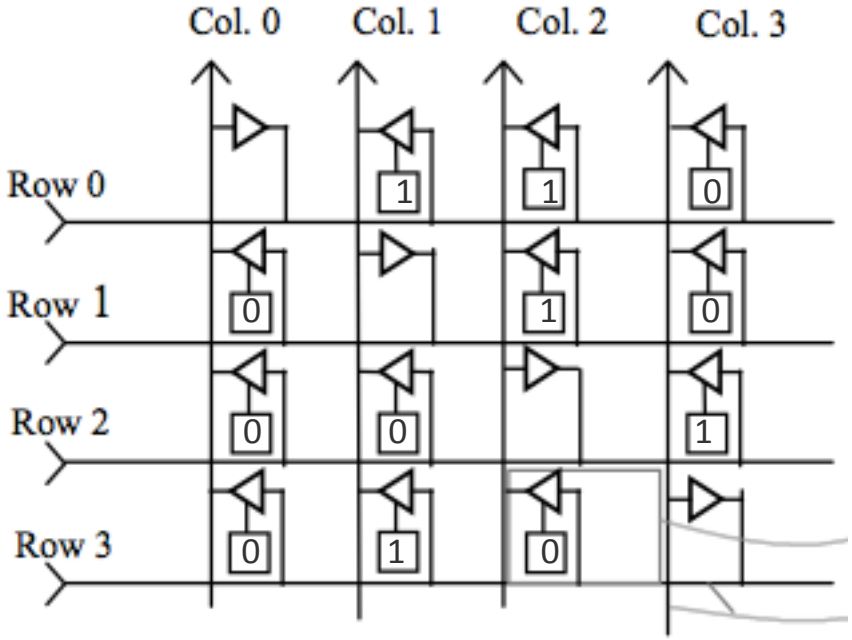
- Dataflow Engines can accelerate PSPI
- Speedup factor = 35x
  (MAX3 vs Intel Westmere)

* PSPI -Phase Shift Plus Interpolation

But what about Graph algorithms?

Assume the following graph expressed using its connectivity matrix:

	v_0	v_1	v_2	v_3
v_0	0	1	1	0
v_1	0	0	1	0
v_2	0	0	0	1
v_3	0	1	0	0



Implements the data-structure AND the algorithm in space (on the HW surface)
Very efficient for operations like: *insert*, *delete*, *reachability*, *shortest path*, …

In conventional system this is heavily dependent on the memory latency and requires data structures, pointers, etc.
For example "reachability" speedup over PC: 10× to 1000× for 1K nodes

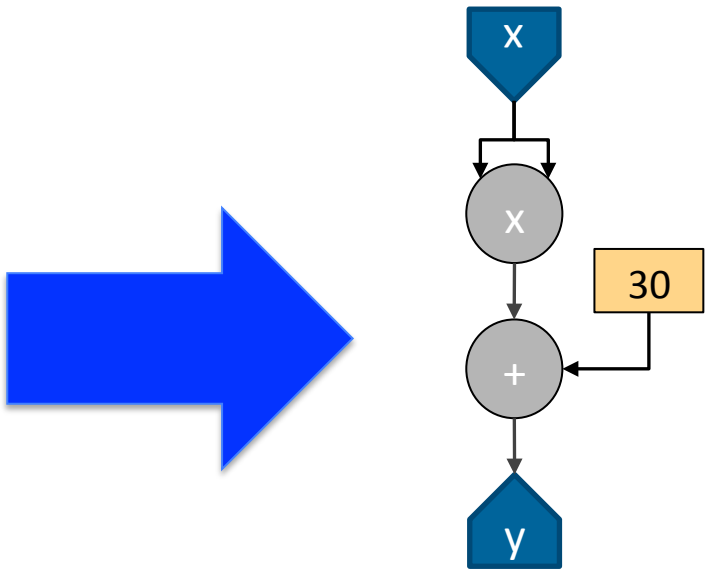
[O. Mencer, Z. Huang, L. Huelsbergen, HAGAR: Efficient Multi-context Graph Processors, FPL 2002]

Converting Simple Expression

$$y_i = x_i \times x_i + 30$$

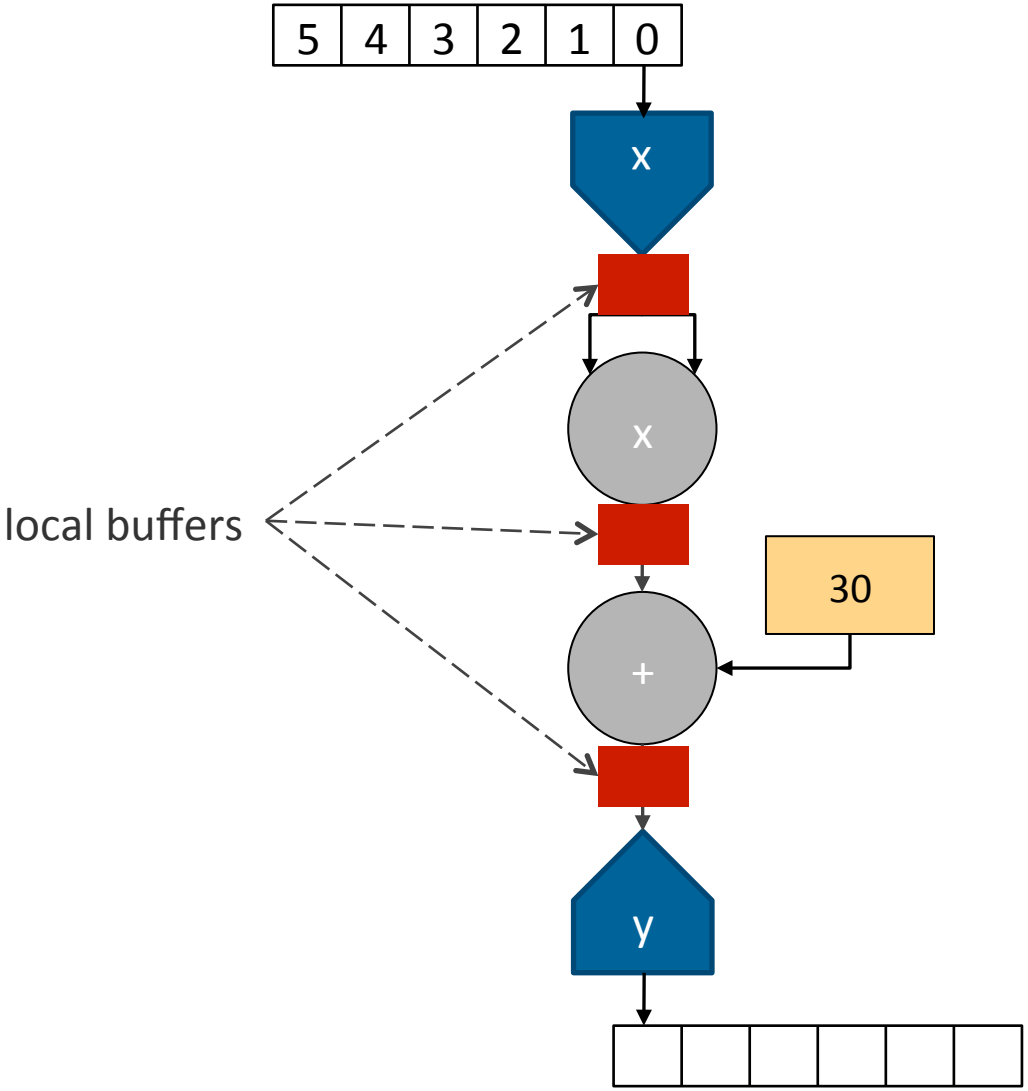
```
int*x, *y;
for (int i =0; i < DATA_SIZE; i++)
    y[i]= x[i] * x[i] + 30;
```

Input stream of integer elements ‘x’

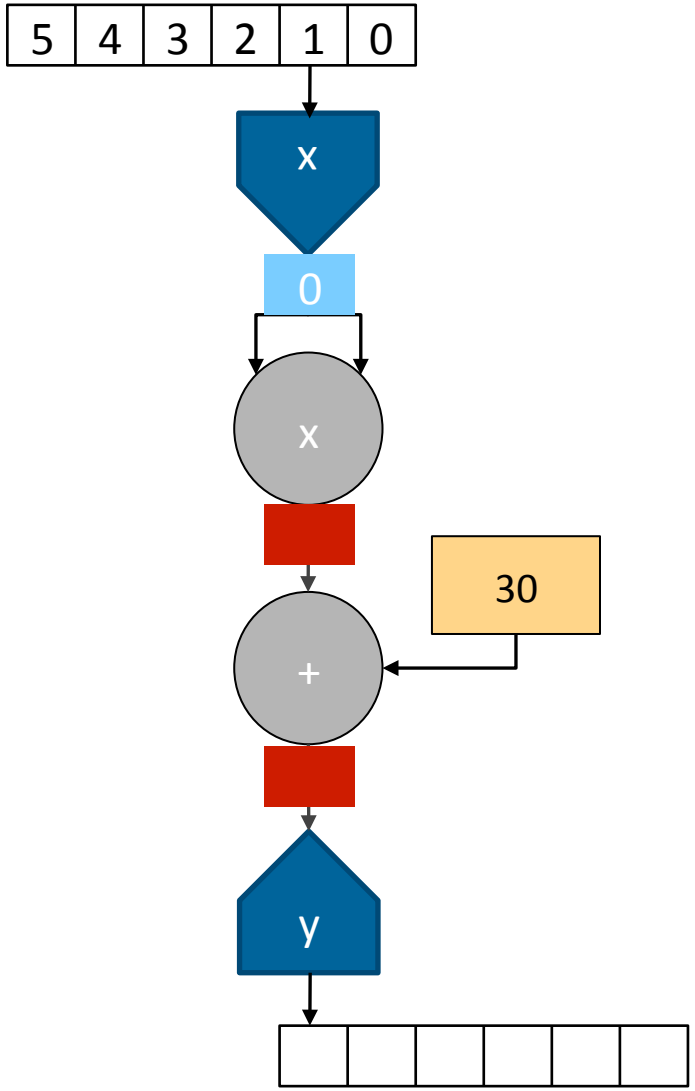


Output stream of integer elements ‘y’

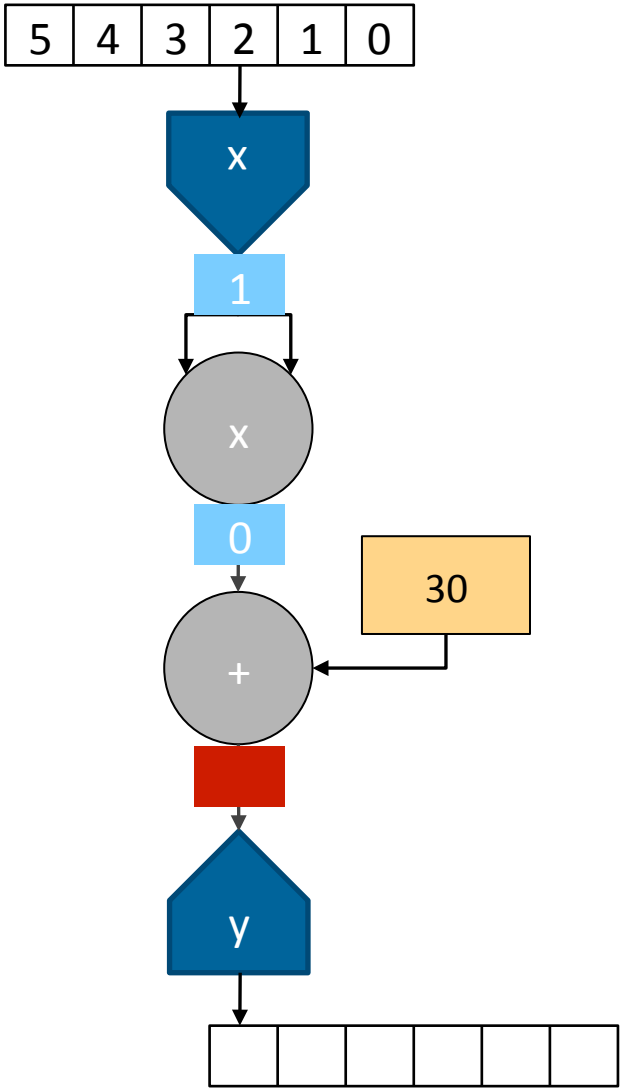
Flowing elements



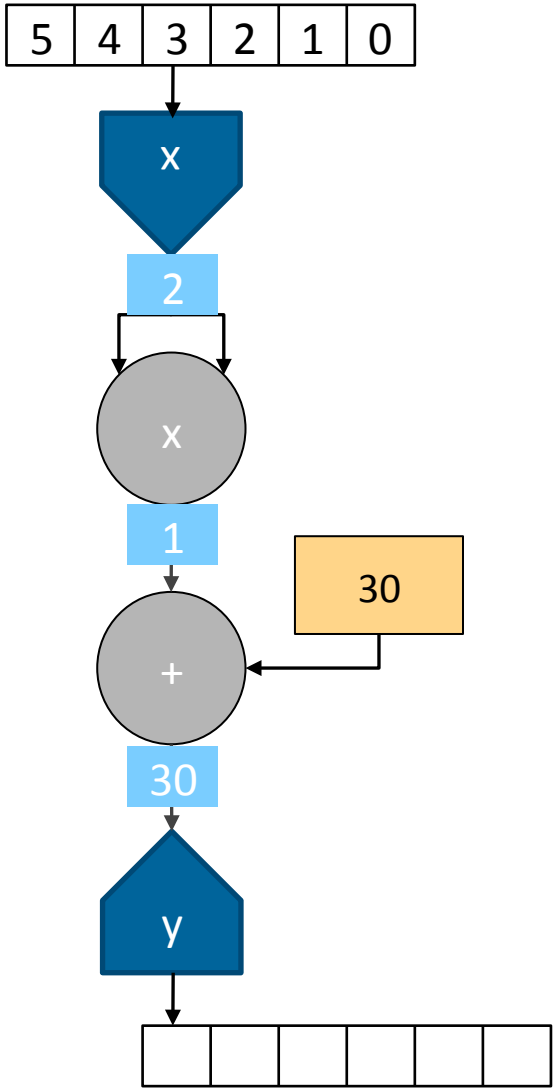
Flowing elements



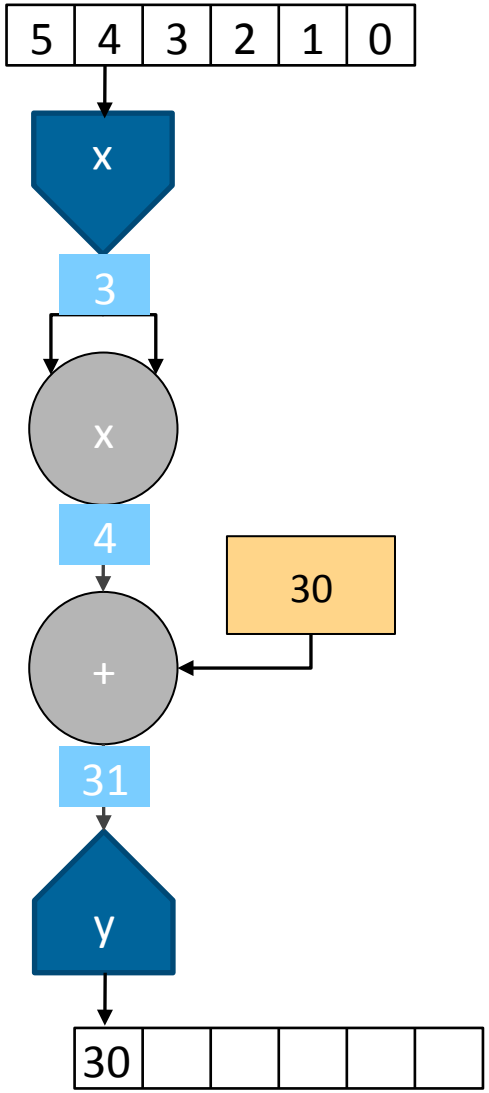
Flowing elements



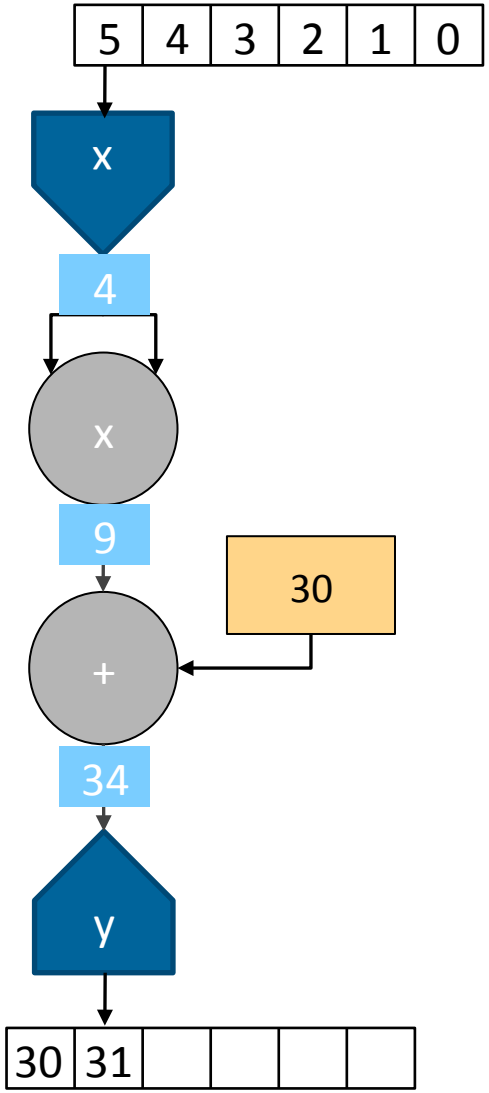
Flowing elements



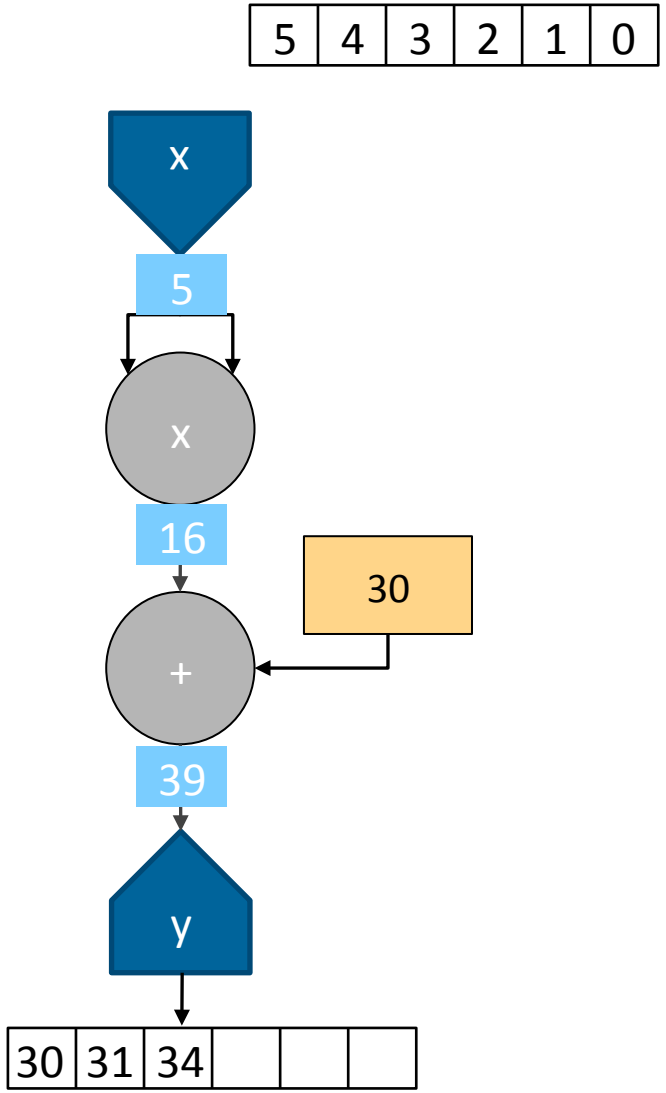
Flowing elements



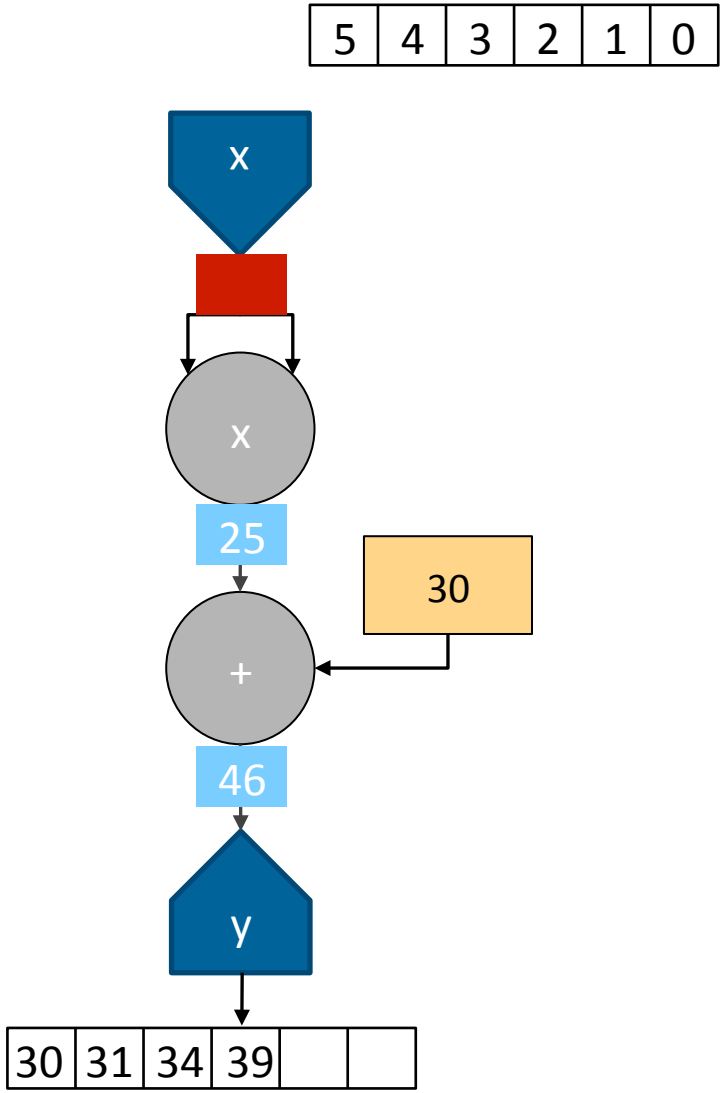
Flowing elements



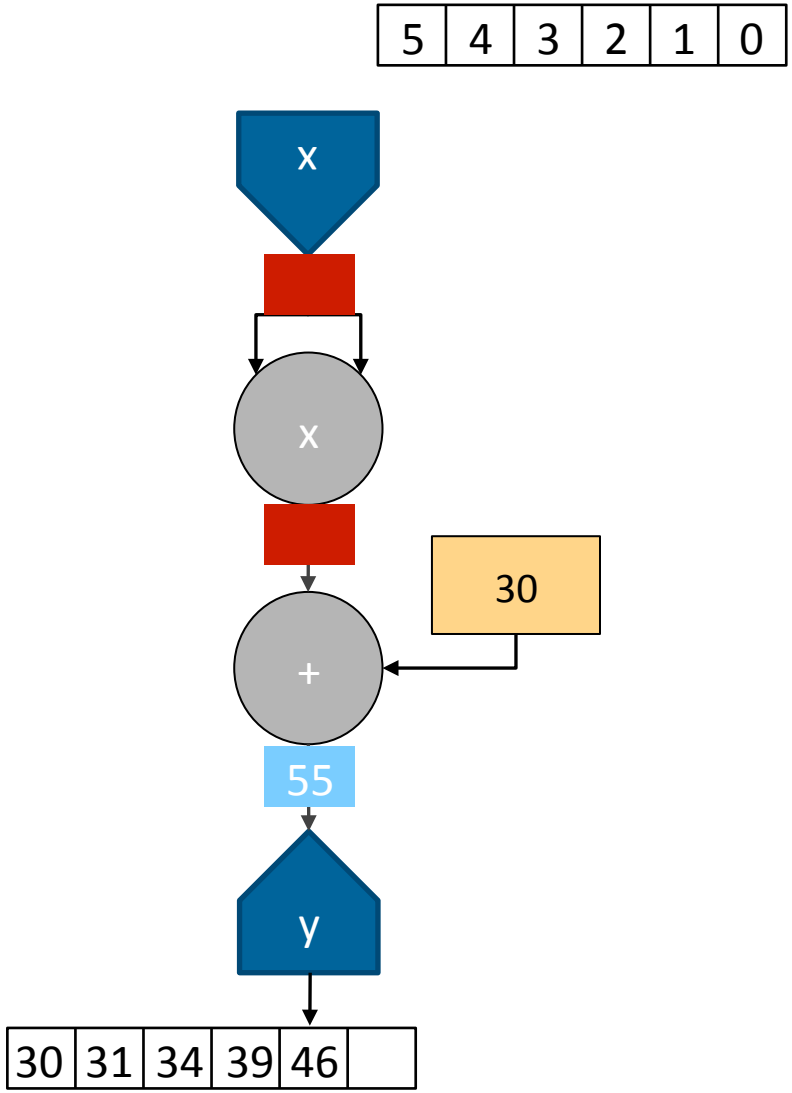
Flowing elements



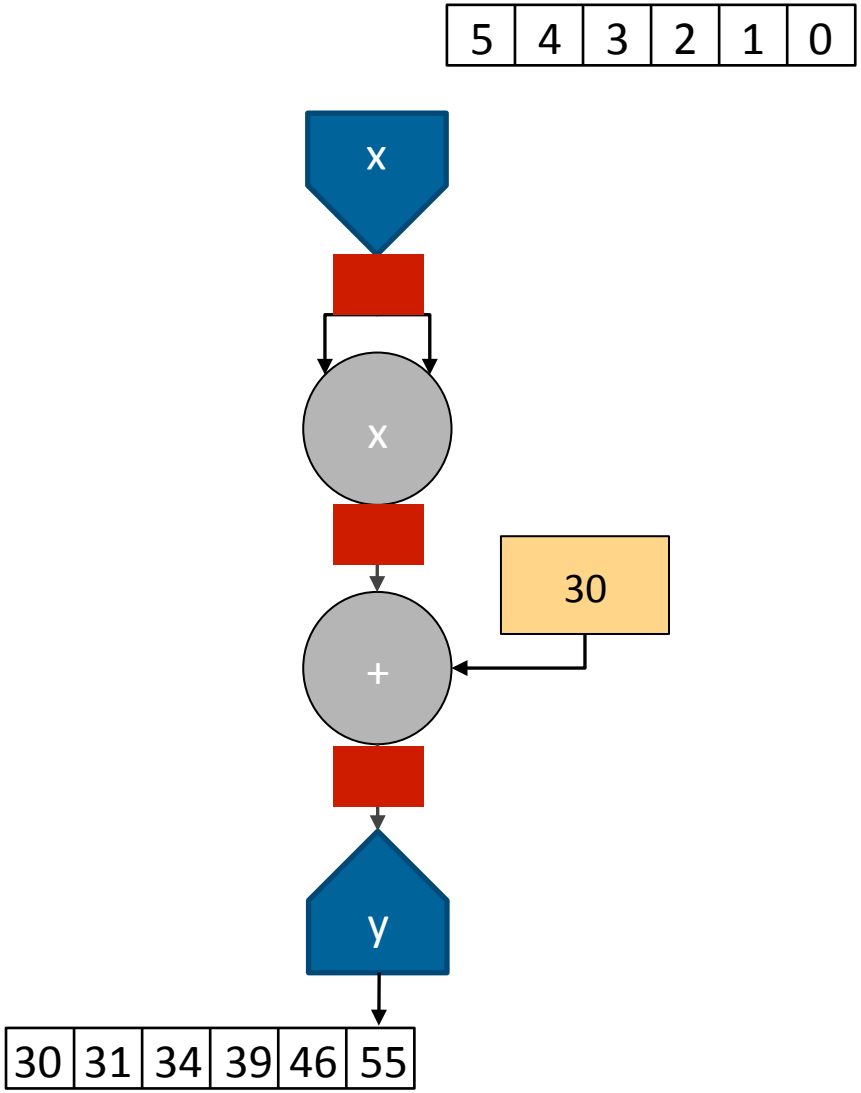
Flowing elements



Flowing elements

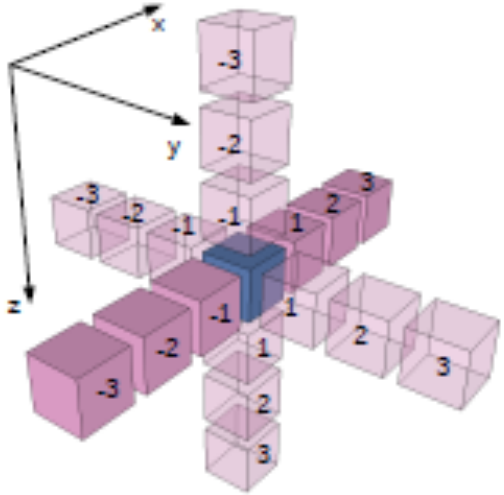


Flowing elements

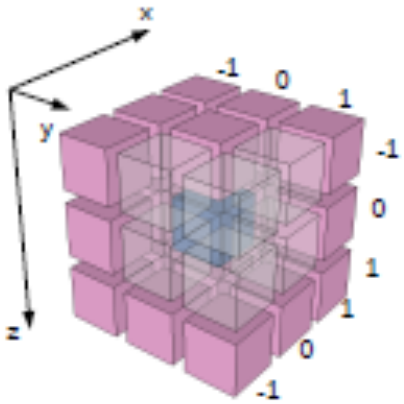


But data can have multiple dimensions

3D Finite Difference Coefficients



(a) 7-point star stencil



(b) 3x3x3 cube stencil

19 MADDs

27 MADDs

Local Buffer = 6 slices

Local Buffer = 3 slices

How to express all of this?

- We need streaming type of variables
- We should be able to access multi-dimensional data
- On the boundaries with the main memory:
  - Address generation
  - Address coalescing
  - Address contention avoidance
  - Memory channels parallelism management
- Local buffers management
- Loop “balancing”
- Data movements minimization
- …

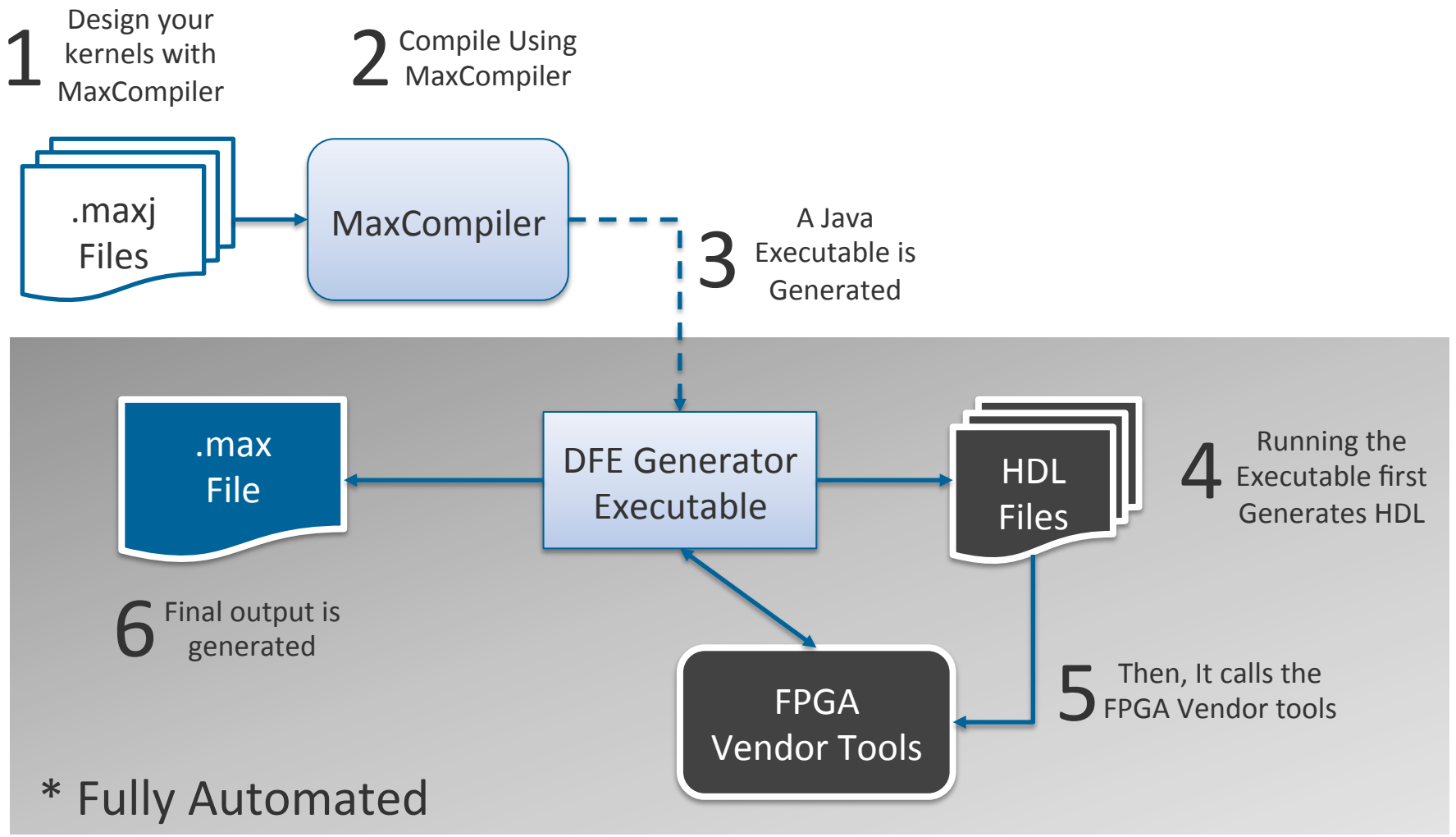
Is there an optimal solution

- Yes but it depends on the application / dataset
- More specifically it depends on the systems “Bottleneck” for the application
- Possible Bottlenecks:
  - Memory access latency
  - Memory access bandwidth
  - Memory capacity
  - Processor local memory size
  - Processor ALU resource
  - Processor ALU operation latency
  - Interconnect bandwidths

How could we can program in space

- Schematic entry of circuits
- Traditional Hardware Description Languages
  - VHDL, Verilog, SystemC.org
- Object-oriented languages
  - C/C++, Python, Java, and related languages
- Functional languages: e.g., Haskell
- High level interface: e.g., Mathematica, MatLab
- Schematic block diagram e.g., Simulink
- Domain specific languages (DSLs)

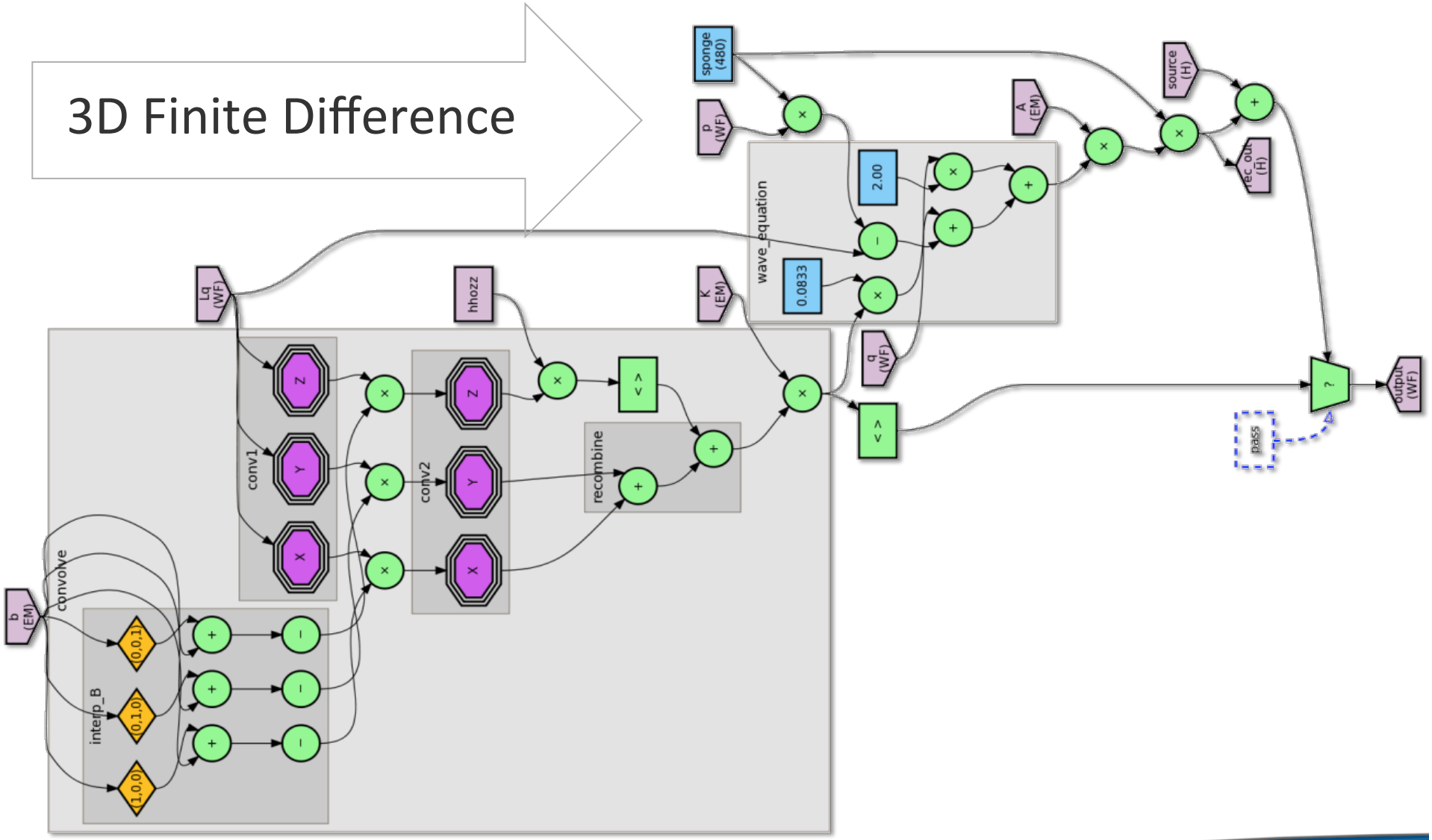
From Graphs to Hardware (Maxeler)



Java meta-programming (Maxeler)

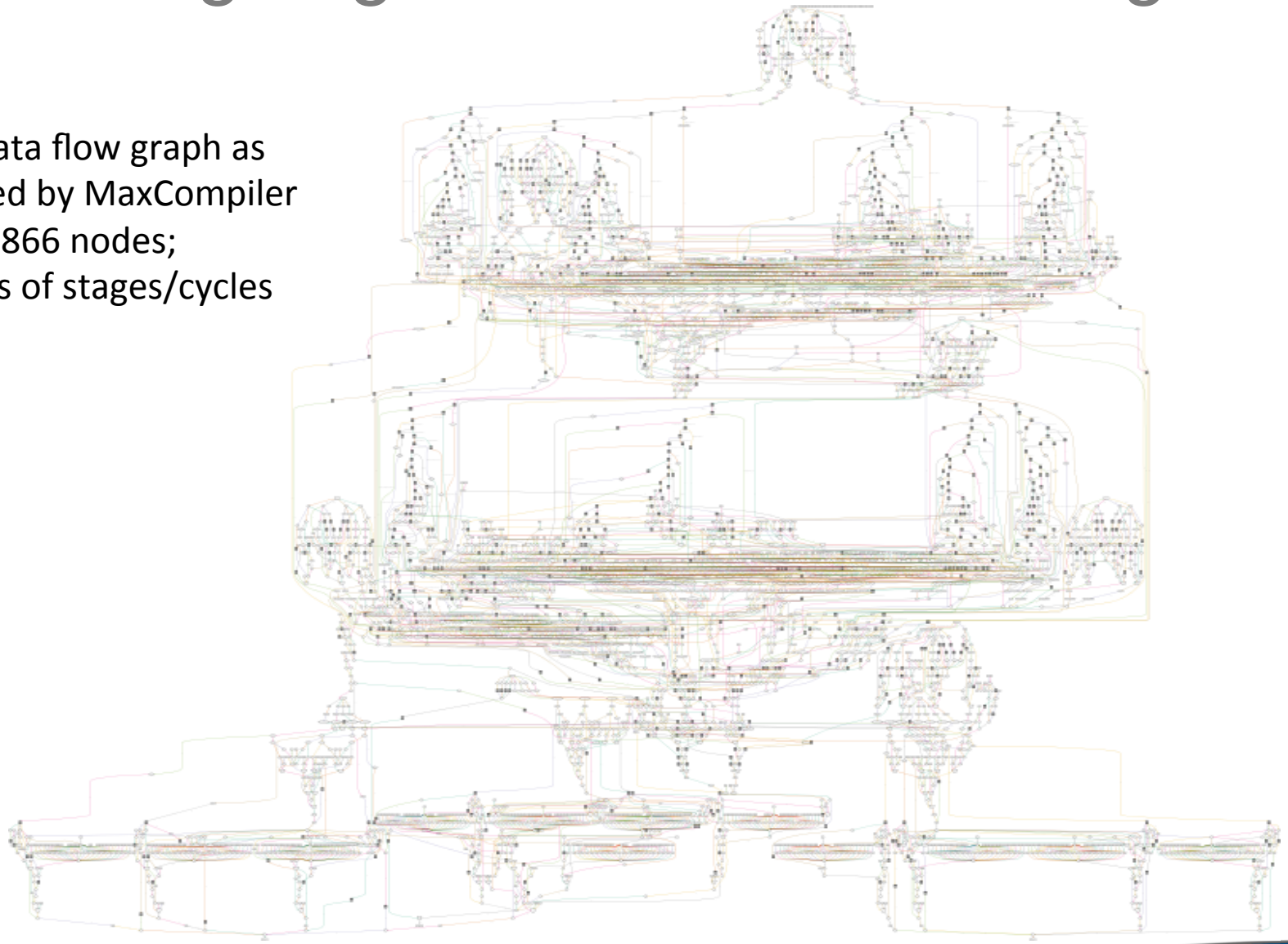
- You can use the full power of Java to write a program that *generates* the dataflow graph
- Java variables can be used as constants in hardware
  - int y; DFEVar x; x = x + y;
- Hardware variables can not be read in Java!
  - Cannot do: int y; DFEVar x; y = x;
- Java conditionals and loops choose *how* to generate hardware → not make run-time decisions
- We will come back later on this

FD multidimensional streaming implementation



Enabling large scale dataflow designs

Real data flow graph as generated by MaxCompiler
4866 nodes;
10,000s of stages/cycles



Summary

- Mathematical expressions are multi-dimensional
- “Fixing” the computational structure and flowing the data through it brings simplicity
- Data access should closely match data structures organization
- There is a practical need to conveniently express very complex and large dataflow graphs
- OpenSPL is here to help with the above