CO405H

Computing in Space with OpenSPL

Topic 3: OpenSPL

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/openspl16/
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

- Introduction to the Basics
- Architecture
- Spatial Substrates
- Control flow in Space

OpenSPL to drive Computing in Space

Dec 10, 2013. London. OpenSPL is being announced at the Bloomberg Enterprise Technology Summit in London. CME, Juniper, Chevron and Maxeler are forming the founding team of companies to drive OpenSPL in their respective markets. Founding academic members are Stanford, Imperial College London, Tsinghua University in Beijing and the University of Tokyo.

Dec 10, 2013. London. The OpenSPL consortium is announcing the first OpenSPL Summer School in July 2014, at Imperial College London. The Summer School will bring together researchers, students, and members of OpenSPL to share experiences, application development, and continue to develop the science behind computing in space.

Dec 9, 2013. Chicago. Ari Studnitzer at CME Group has published a blog on OpenMarket about the announcement of OpenSPL and OpenSPL-compliant technology from the CME. "Our new, patent-pending iLink Market Segment gateway will provide inline pre-trade credit controls, unparalleled predictability, advanced market controls, as well as increased capacity and throughput."

- Founding Corporations:

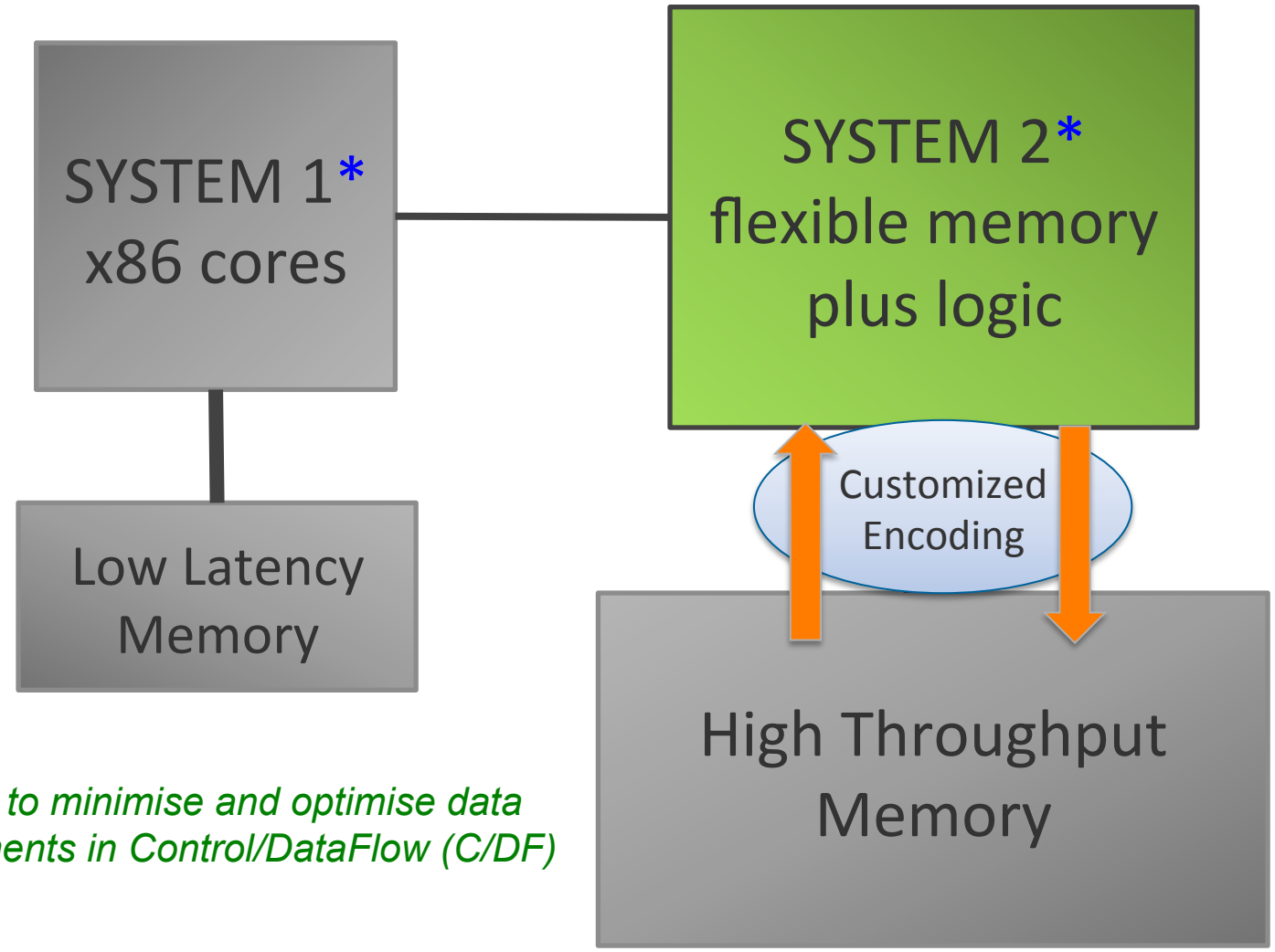
Chevron Human Energy · CME Group · JUNIPER NETWORKS · MAXELER Technologies MAXIMUM PERFORMANCE COMPUTING

- Academic Partners:

Imperial College London · Stanford · 東京大学 THE UNIVERSITY OF TOKYO · 清華大學 Tsinghua University

OpenSPL http://www.OpenSPL.org launched on Dec 9, 2013

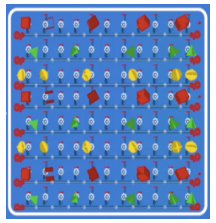
The Combined Control/DataFlow system



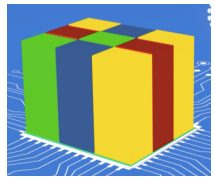
Goal is to minimise and optimise data movements in Control/DataFlow (C/DF)

*** System 1 and System 2 are based on D Kahneman, "Thinking Fast Thinking Slow", Nobel Prize in Economics, 2002**

OpenSPL basics



- Control and Data-flows are decoupled
  - Both are fully programmable
- Operations exist in space and by default run in parallel
  - Their number is limited only by the available space
- All operations can be customized at various levels
  - e.g., from algorithm down to the number representation
- Multiple operations constitute kernels
- Data streams through the operations / kernels
- The data transport and processing can be balanced
- All resources work all of the time for max performance
- The In/Out data rates determine the operating frequency

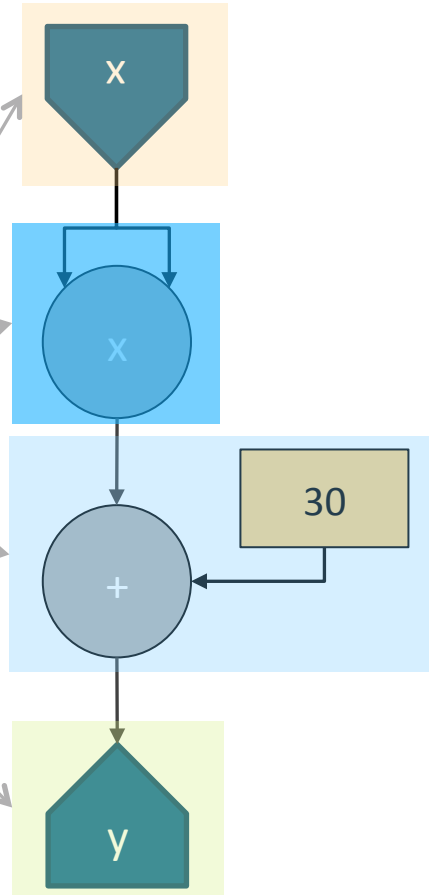


Equally spread the available “forces” and move no faster than required by the application

OpenSPL example: $X^2 + 30$

Syntax Directed Translation

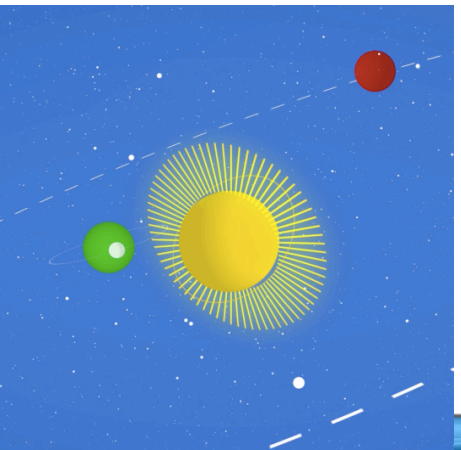
```
SCSVar x = io.input("x", scsInt(11));

SCSVar result = x * x + 30;

io.output("y", result, scsInt(23));
```

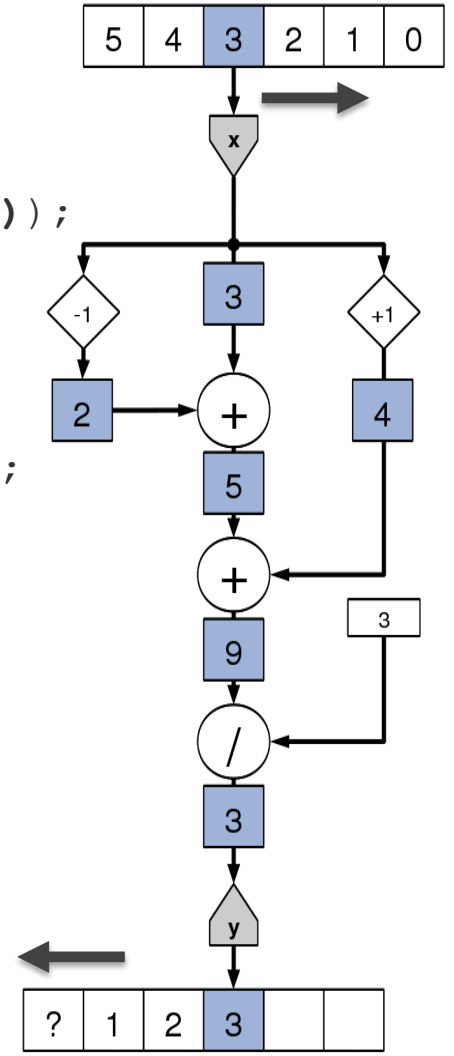


OpenSPL

OpenSPL example: Moving Average

```
class MovingAvgKernel extends Kernel {
    MovingAvgKernel() {
        SCSVar x = io.input("x", scsFloat(7,17));
        SCSVar prev = stream.offset(x, -1);
        SCSVar next = stream.offset(x,  1);
        SCSVar sum = prev + x + next;
        SCSVar result = sum / 3;
        io.output("y", result, scsFloat(7,17));
    }
}
```

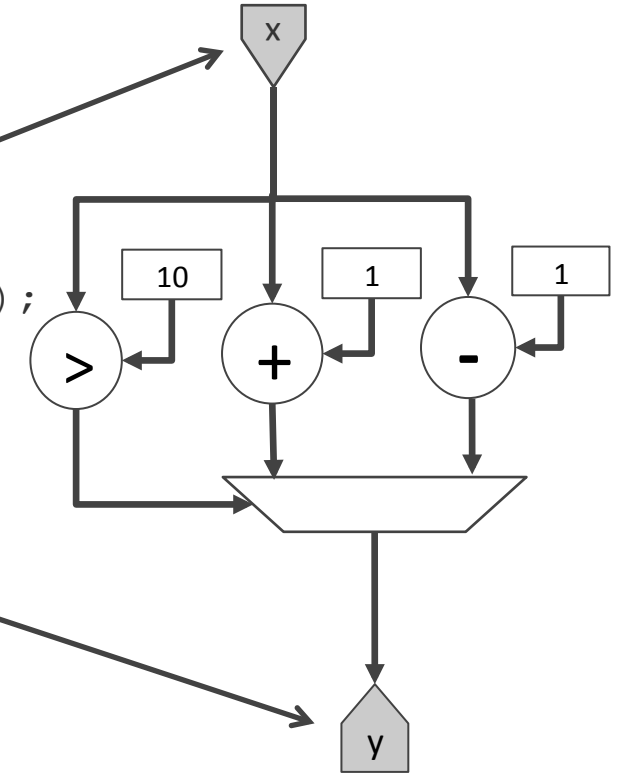


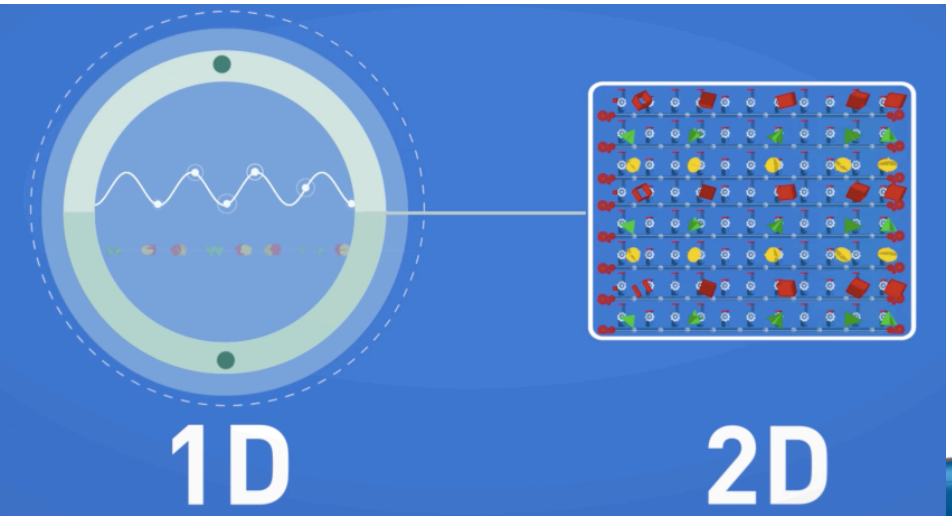


OpenSPL example: Control in Space

or What do we do with IF statements

```
class SimpleKernel extends Kernel {
    SimpleKernel() {
        SCSVar x = io.input("x", scsFix(24));
        SCSVar result = (x>10) ? x+1 : x-1;
        io.output("y", result, scsFix(25));
    }
}
```





The Machine Model of Space

- A Spatial Computing system consists of:
  - a Chip: a *Spatial Computing Substrate* (SCS): hardware technology with flexible arithmetic units and programmable interconnect
  - SCS specific compilation toolchain
  - SCS specific runtime system and all low level software
- Three basic memory types:
  - Scalars
  - Fast Memory (FMEM): small and fast
  - Large Memory (LMEM): large and slow

Fast and Slow

John von Neumann, 1946:

“We are forced to recognize the possibility of constructing a hierarchy of memories, each of which has greater capacity than the preceding, but which is less quickly accessible.”

The Machine Model of Space

- Computational kernels interconnected by data flow streams to form bigger entities called *actions*
- Action is the basic the Spatial Computing Substrate (SCS) execution unit and performs as a single entity
- In a spatial system one or more SCS engines exist, each executing a single action at any moment in time

Arithmetic in Space

- Operations instantiated as separate arithmetic units
- Units along data paths use custom arithmetic and number representation (as long data stays correct)
- The above may reduce individual unit sizes (and maximizes the number that fits a given SCS)
- Data rates of memory and I/O communication may also be maximized due to scaled down data sizes

Multiscale Arithmetic in Space

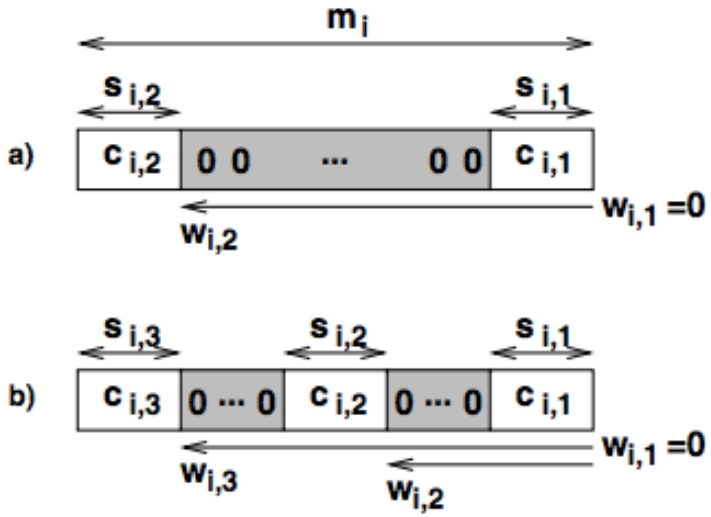
- Arithmetic optimizations at the **bit level**
  - e.g., minimizing the number of ’1’s in binary numbers, leading to linear savings of both space and power (the zeros are omitted in the implementation)
- **Higher level arithmetic** optimizations
  - e.g., in matrix algebra, the location of all non-zero elements in sparse matrix computations is important
- Spatial **encoding of data** structures can reduce transfers between memory and computational units (boost performance and improve efficiency)
  - In temporal computing encoding and decoding would take time and eventually can cancel out all of the advantages
  - In spatial computing, encoding and decoding just consume a bit more of additional space

Multiscale Optimisations

Multiple scales of computing	Important features for optimization
complete system level	⇒ balance compute, storage and IO
parallel node level	⇒ maximize utilization of compute and interconnect
microarchitecture level	⇒ minimize data movement
arithmetic level	⇒ tradeoff range, precision and accuracy = discretize in time, space and value
bit level	⇒ encode and add redundancy
transistor level	=> create the illusion of ‘0’ and ‘1’

And more, e.g., trade Communication (Time) for Computation (Space)

Expl: Minimize ‘1’s => Sparse Coefficients



$$p = \frac{32799}{32768} - \frac{609}{32768}x - \frac{14881}{32768}x^2.$$

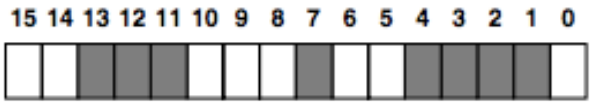


Fig. 2. Target format for cos function.

$$p = -\frac{75}{32768} + \frac{34538}{32768}x - \frac{6169}{32768}x^2.$$

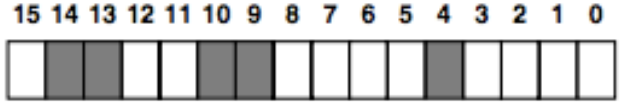


Fig. 3. Target format for sin function.

$$p = \frac{32793}{32768} + \frac{31836}{32768}x + \frac{21146}{32768}x^2.$$

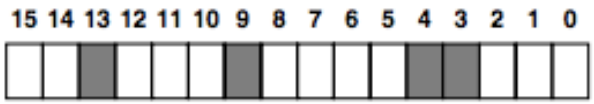


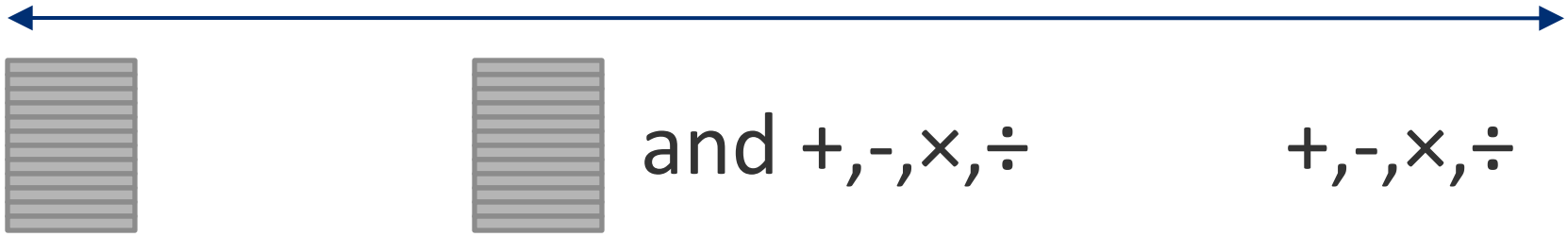
Fig. 4. Target format for exp function.

Nicolas Brisebarre, Jean-Michel Muller and Arnaud Tisserand
Sparse Coefficient Polynomial Approximations for Hardware Implementation,
Asilomar Conference, 2004.

To compute or remember, that is the question.

Computing f(x) in the range [a,b] with $|E| \leq 2^{-n}$

Table *Table+Arithmetic* *Arithmetic*



- uniform vs non-uniform
- number of table entries
- how many coefficients

- polynomial or rational approx
- continued fractions
- multi-partite tables

Underlying hardware/technology changes the optimum

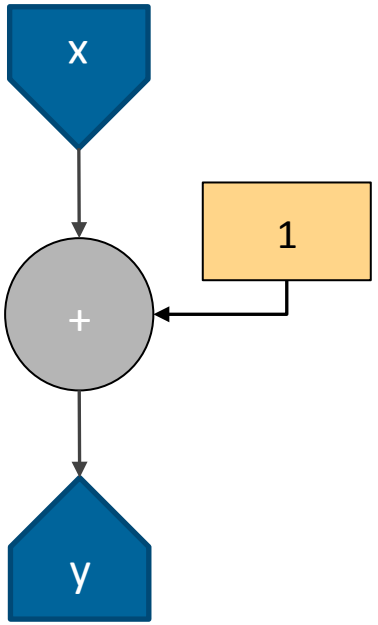
Meta-programming

- You can use the full power of your language of choice, e.g., Java, to write a program that *generates* the computation in space
- Variables can be used as constants in hardware (actually wires)
  - int y; SCSVar x; x = x + y;
- Hardware variables do not "exist" at compile time!
  - Cannot do: int y; SCSVar x; y = x;
- Conditionals and loops choose *how* to generate hardware → not make run-time decisions

Runtime vs Compiletime Variables

What dataflow graph is generated?

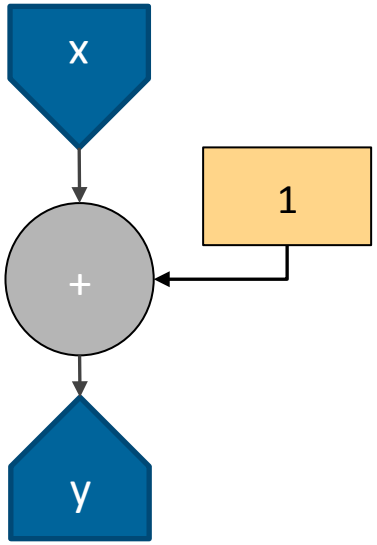
```
SCSVar x = io.input("x", <type>);
SCSVar y;
int CON = 1;

y = x + CON;

io.output("y", y, <type>);
```



Runtime versus Compiletime Variables (cont)

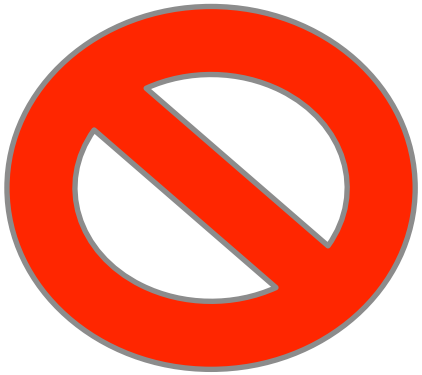
What dataflow graph is generated?

```
SCSVar x = io.input("x", <type>);
int s = 10;
SCSVar y;

if (s < 100) { y = x + 1; }
else { y = x - 1; }

io.output("y", y, <type>);
```



What dataflow graph is generated?

```
SCSVar x = io.input("x", <type>);
SCSVar y;

if (x < 10) { y = x + 1; }
else { y = x - 1; }

io.output("y", y, <type>);
```

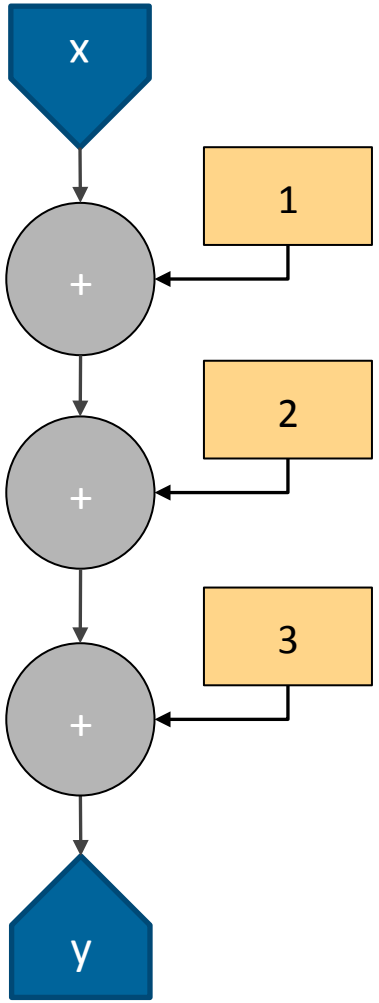


Compile error.
You can't use the value of stream 'x' in a meta conditional

Loops

What dataflow graph is generated?

```
SCSVar x = io.input("x", <type>);
SCSVar y = x;
for (int i = 1; i <= 3; i++) {
        y = y + i;
}
io.output("y", y, <type>);
```



Can make the loop any size – until you run out of space on the chip!
Larger loops can be partially unrolled in space and reused multiple times in time

If-Else Statements

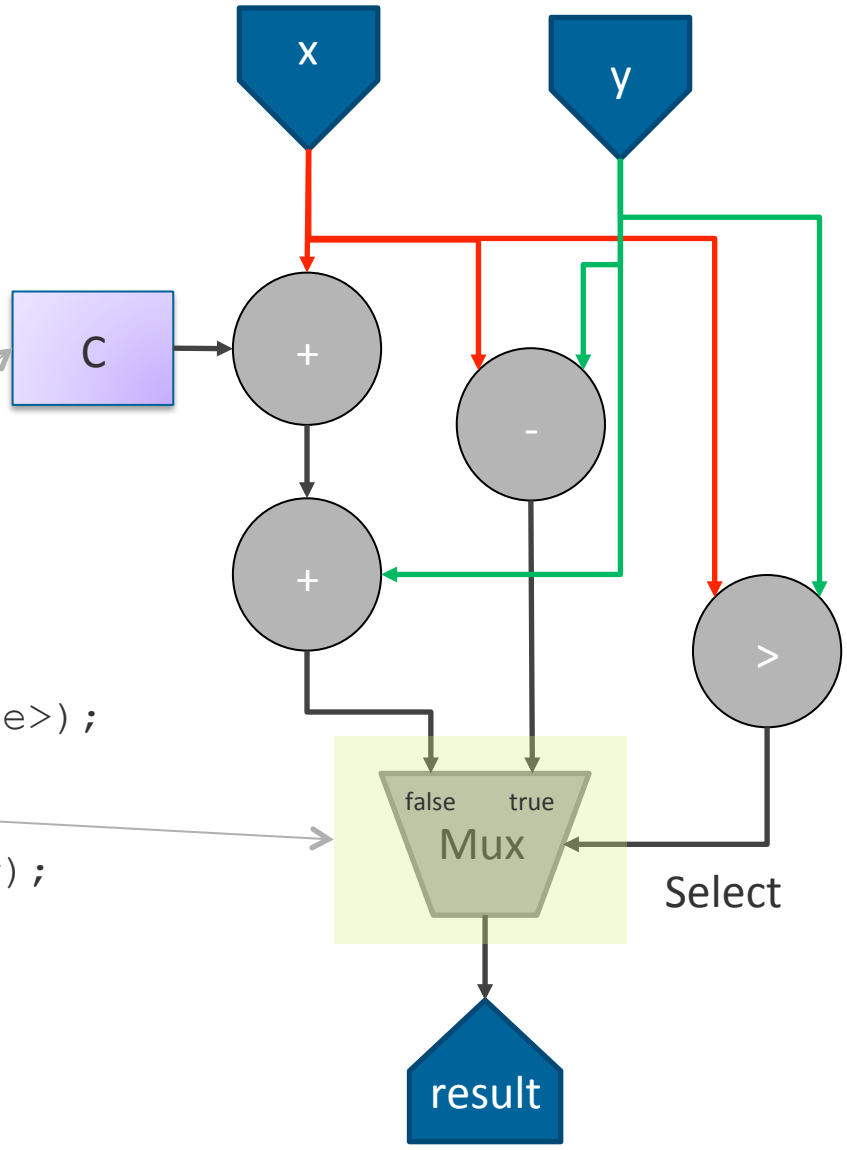
- Data dependent conditional statements are common
- How can we implement this in OpenSPL?

```
int C = 500;
for (int i = 0; i < N; i++) {
     if (x[i] > y[i])
        result[i] = x[i] – y[i];
     else
        result[i] = C + x[i] + y[i];
}
```

Control Flow in Space
If-Else = Mux

What is this scalar input?

```
SCSVar x = io.input("x", <type>);
SCSVar y = io.input("y", <type>);
SCSVar C = io.scalarInput("C", <type>);

SCSVar result = x > y ?
                (x - y) : (C + x + y);
```



Scalar Inputs Connect Space and Time

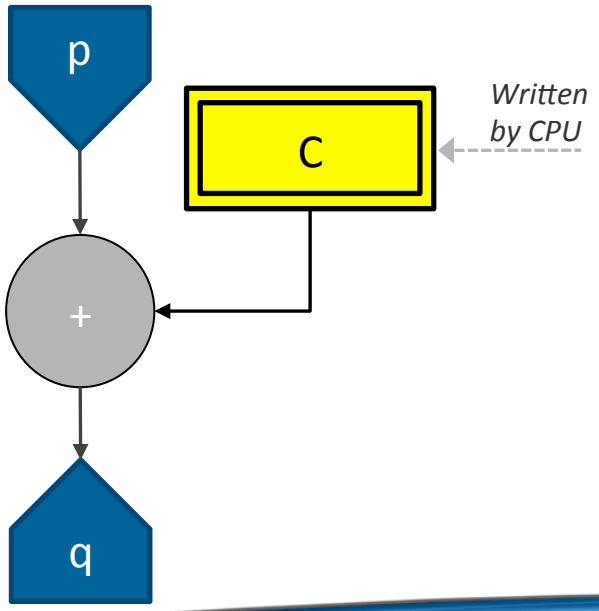
- Consider:

```
void fn1(int N, int *q, int *p) {
    for (int i = 0; i < N; i++)
        q[i] = p[i] + 4;
}
```

```
void fn2(int N, int *q, int *p, int C) {
    for (int i = 0; i < N; i++)
        q[i] = p[i] + C;
}
```

- In fn2, we can change the value of C without recompiling, but it is constant for the whole loop
- OpenSPL equivalent:

```
SCSVar p = io.input("p", scsInt(32));
SCSVar C = io.scalarInput("C", scsInt(32));

SCSVar q = p + C;

io.output("q", q, scsInt(32));
```



Summary

- OpenSPL is an open concept for a language based on computing in space
- OpenSPL provides the core constructs to map computation onto a 2D compute fabric
- OpenSPL concepts can be targetted to different chip platforms, while performance optimisations are highly algorithm and chip specific