High-level programming of data-centric reconfigurable dataflow systems

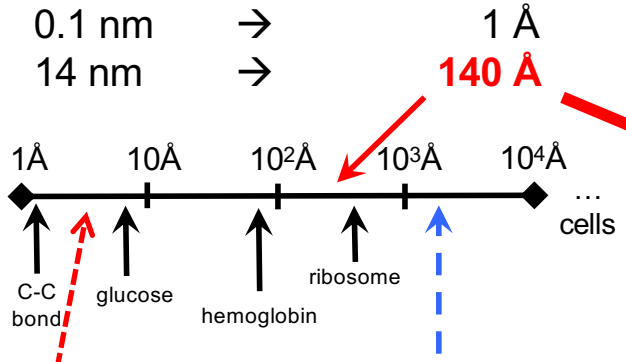
Georgi Gaydadjiev, Director of Maxeler IoT-Labs BV, Delft
Honorary Visiting Professor at the Department of Computing, Imperial College London

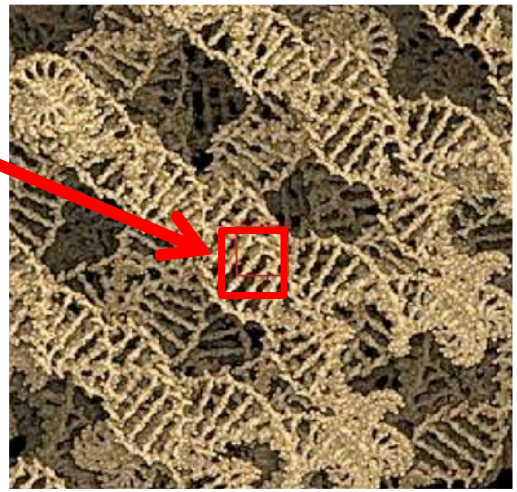
MAXELER Technologies
MAXIMUM PERFORMANCE COMPUTING

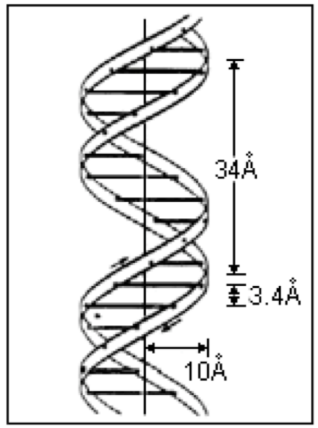
***HLPP 2019**, Linköping University,*
Linköping, Sweden, 4 July 2019

Think Ångströms not nanometers

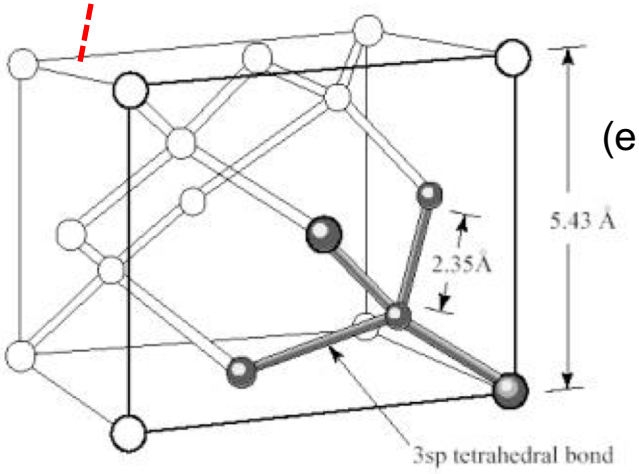
We should steer the movements of almost each individual electron to solve our specific problem





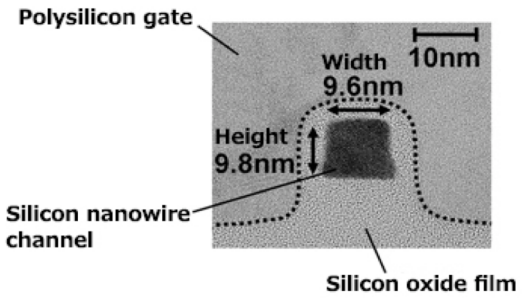


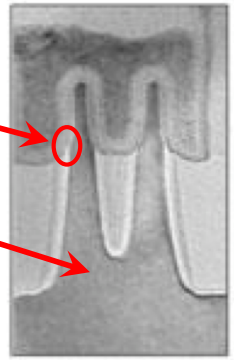
DNA



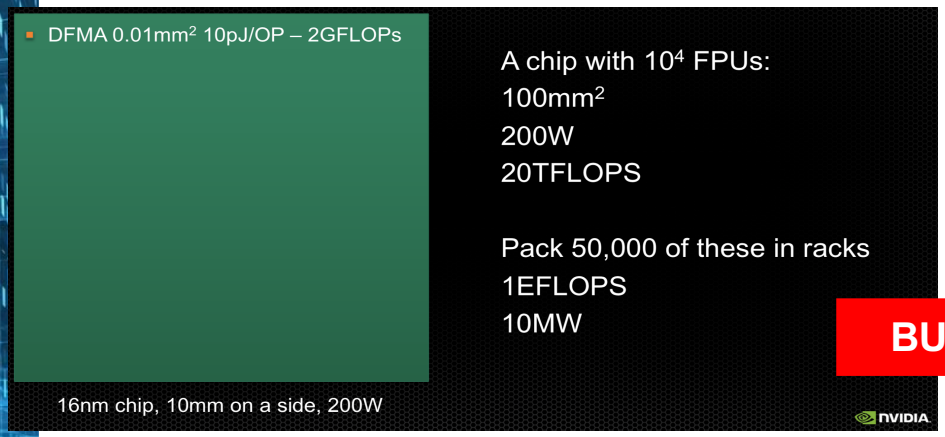
very few atoms
(e.g., 3nm / 30Å → **6 to 12 atoms**)

100s of Si atoms in 14nm





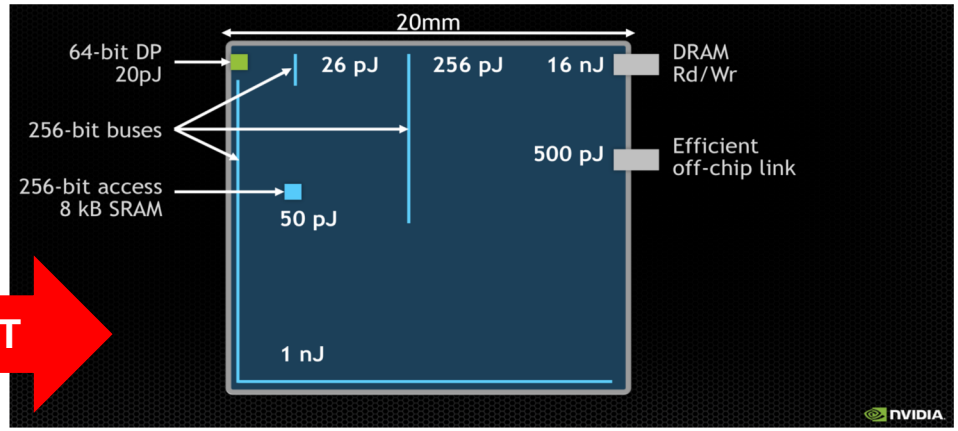
The Data Movement Challenge



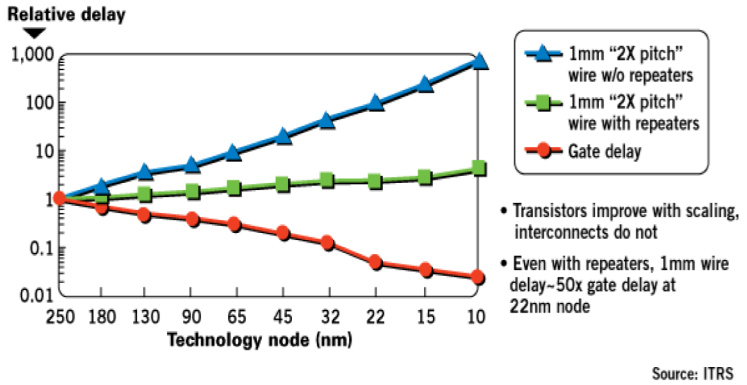
A chip with 10^4 FPUs:
100mm²
200W
20TFLOPS

Pack 50,000 of these in racks
1EFLOPS
10MW

BUT



Processor Technology	40 nm	10nm
Vdd (nominal)	0.9 V	0.7 V
DFMA energy	50 pJ	7.6 pJ
64b 8 KB SRAM Rd	14 pJ	2.1 pJ
Wire energy (256 bits, 10mm)	310 pJ	174 pJ

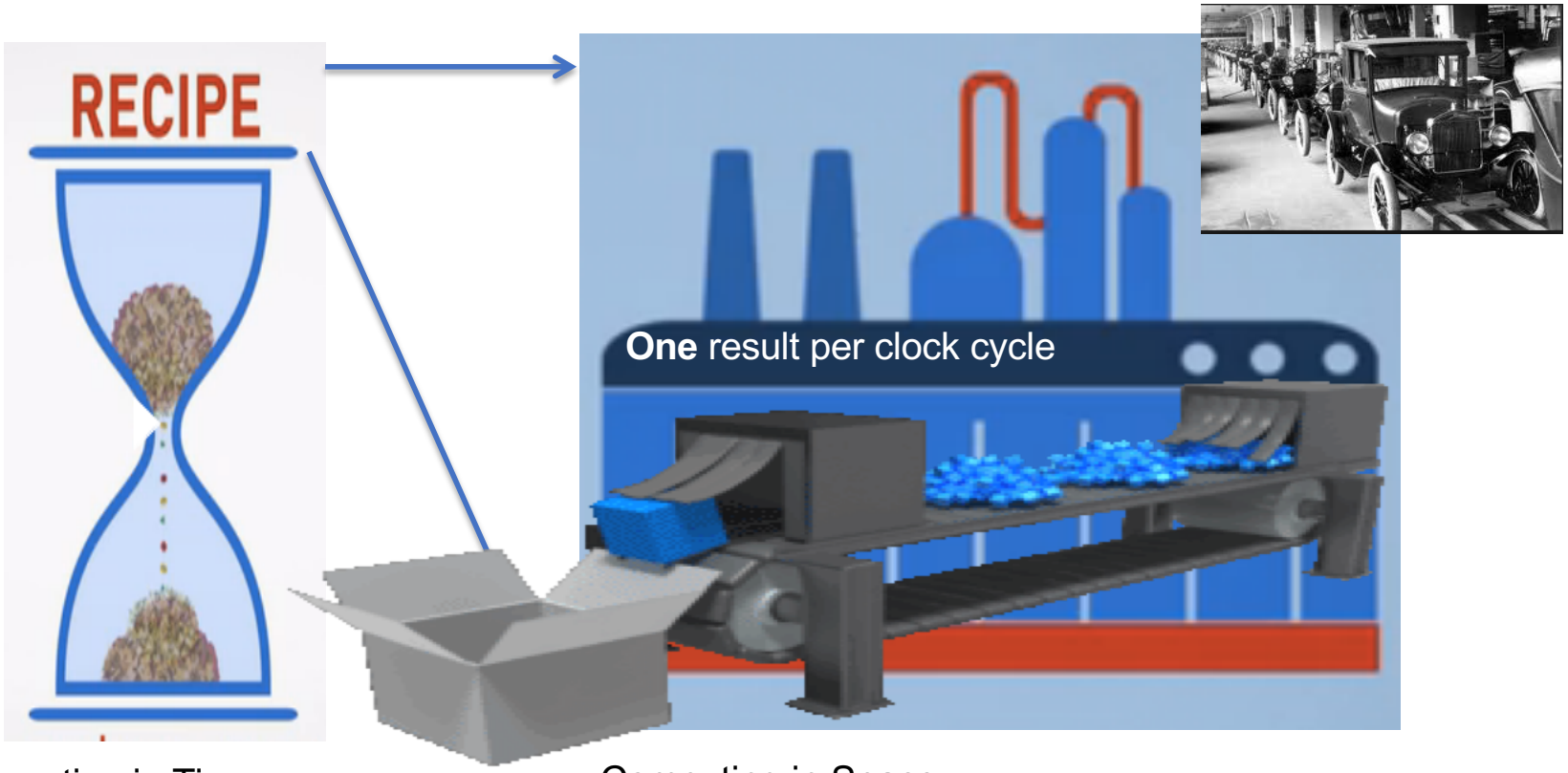


(Courtesy: NVIDIA and ITRS)

Moving data off-chip will cost ~200x more energy and is also much slower

Wires that carry the data (and instructions, if any) at all levels should be considered seriously

Build Computers for your Problem and Data



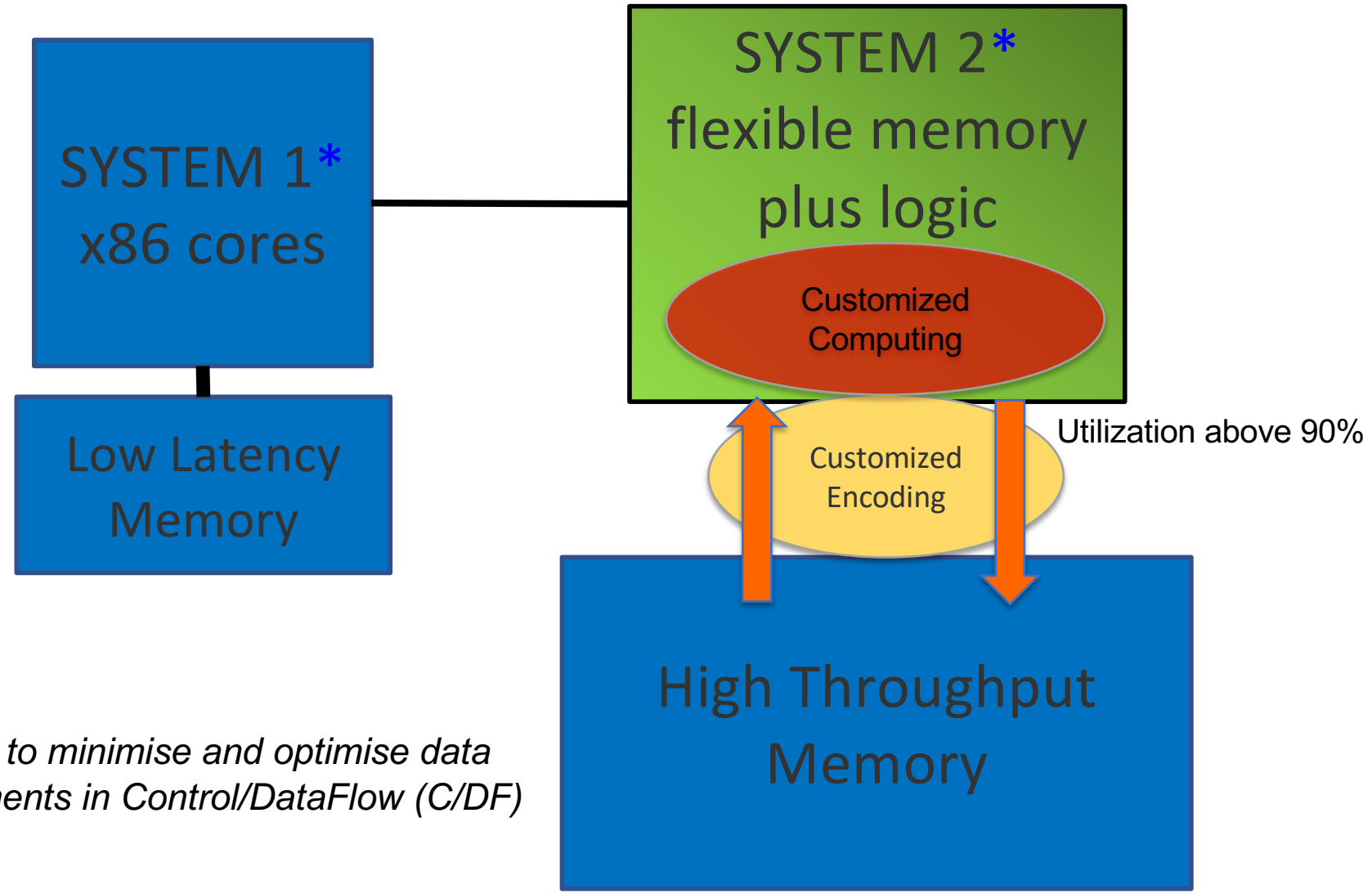
Computing in Time:
Follow a recipe step by step one at the time

Computing in Space:
Build a "recipe specific" factory with multiple paths performed simultaneously

Efficient, predictable, reliable "mass production" of huge data amounts

At each clock tick all data in processing move one stage ahead -> massive throughput

The Combined Control/DataFlow System



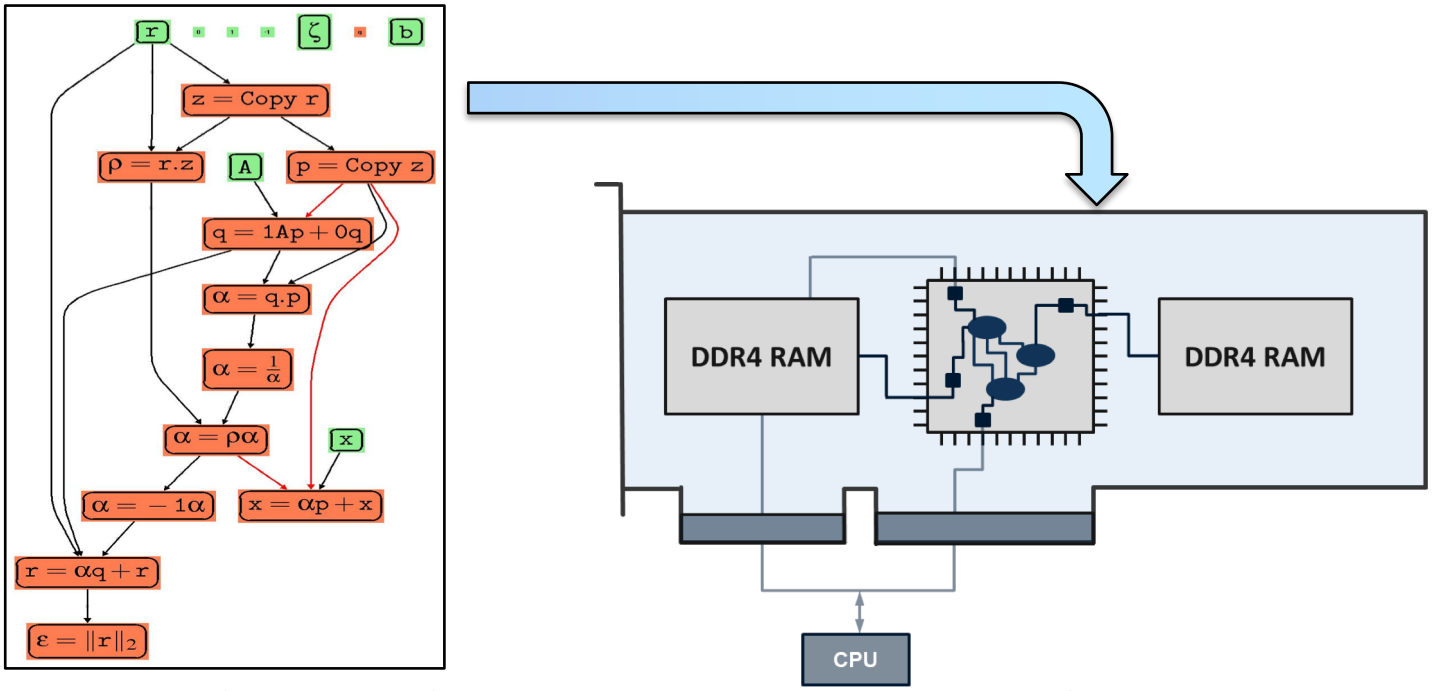
Goal is to minimise and optimise data movements in Control/DataFlow (C/DF)

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

Programming a Dataflow "mass production" Engine

Create customized mega accelerators with massive inherent throughputs

2. Compile dataflow structure and load to hardware



1. Describe Conjugate Gradient as dataflow graph

3. Stream data through the Custom Accelerator

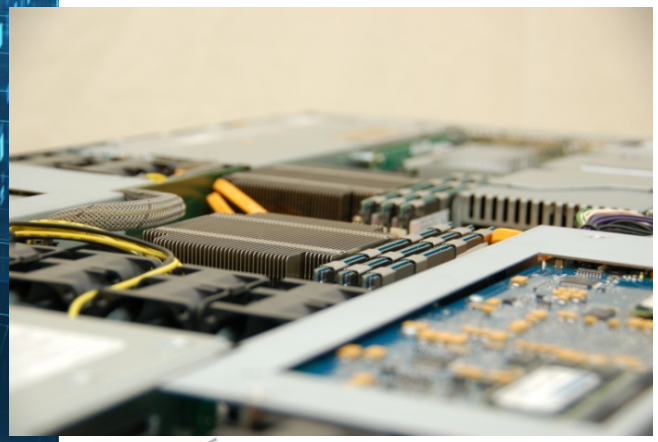
From Equations to Dataflow Hardware

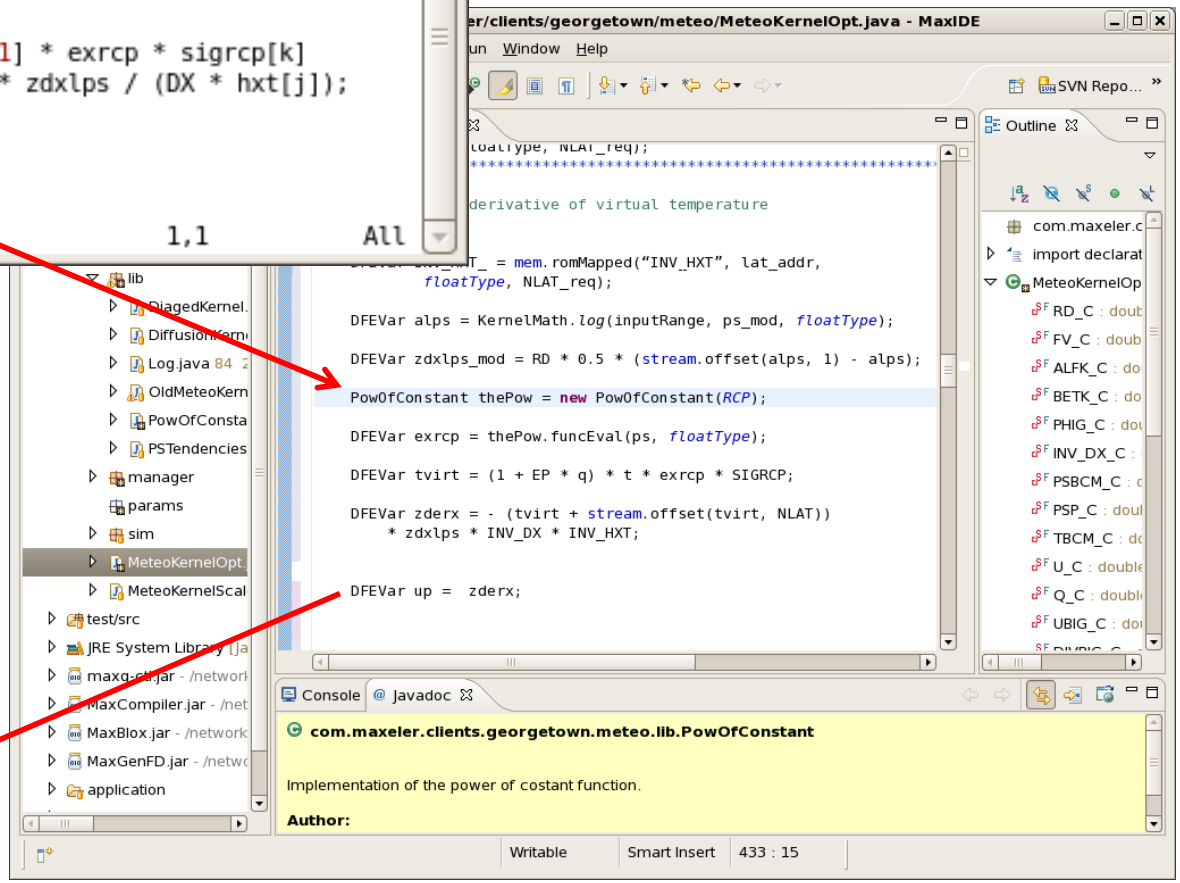
$$\frac{\partial u}{\partial t} = -\frac{u}{ah_x}\frac{\partial u}{\partial \lambda} - \frac{v}{a}\frac{\partial u}{\partial \varphi} - \dot{\sigma}\frac{\partial u}{\partial \sigma} - \frac{R_d T_v}{ah_x}\frac{\partial \ln p_s}{\partial \lambda} - \frac{1}{ah_x}\frac{\partial \Phi}{\partial \lambda} + F_u$$

Vi Editor: meteo.c

File Edit View Terminal Tabs Help

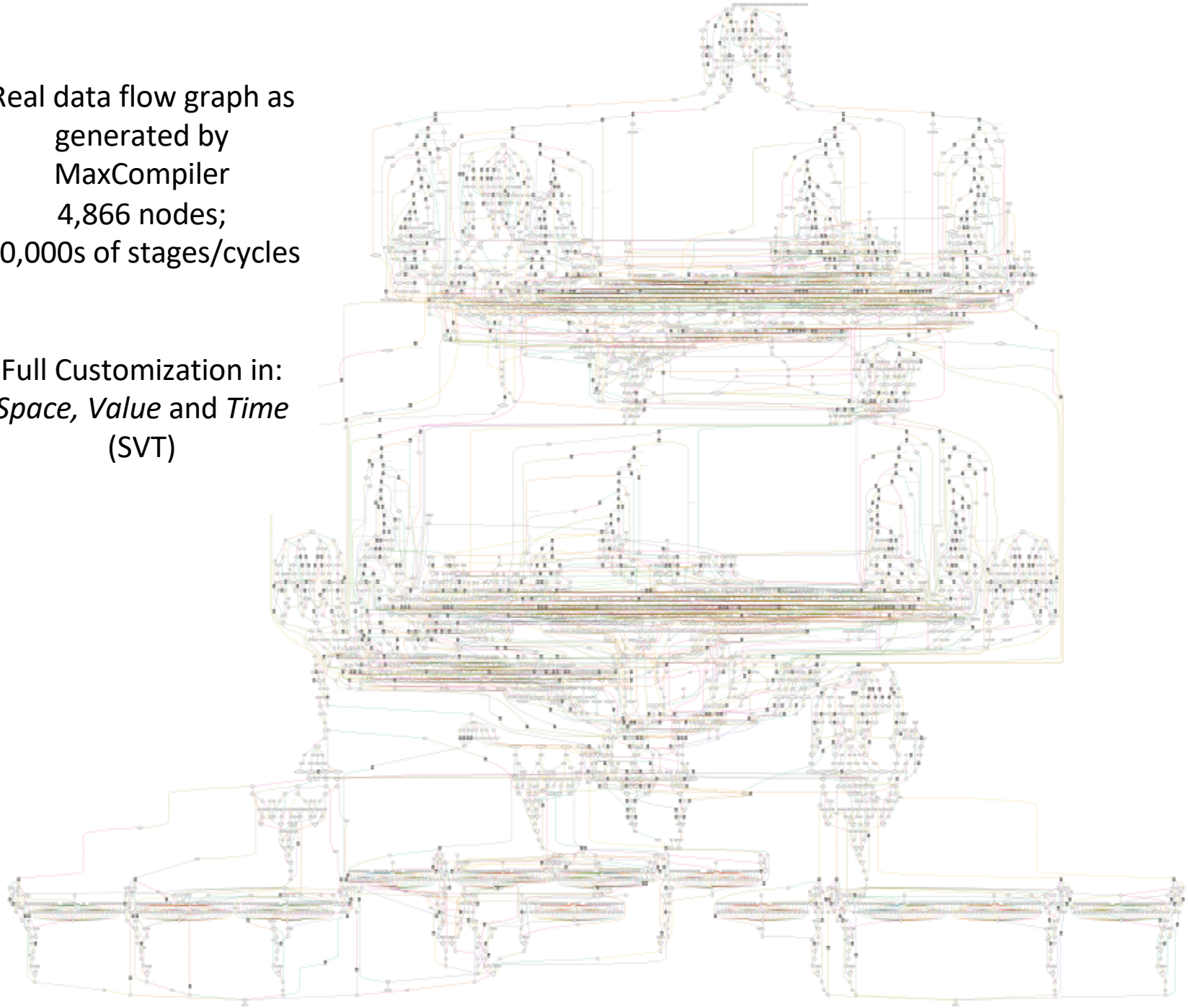
```
/* Compute Virtual Temperature contribution*/
for(int i3D=0; i3D<NLAT*NLON*NLEV; i3D++) {

  float alps = log(ps[i2D]);
  float alps_p1 = log(ps[i2D + 1]);
  float zdxlps = RD * 0.5f * (alps_p1 - alps);
  float exrcp = exp(RCP * alps_p1);
  float tvirt = (1.0f + EP * q[i3D + 1])* t[i3D + 1] * exrcp * sigrcp[k]
  float zderx = - (tvirt[i3D] + tvirt[i3D + NLAT])* zdxlps / (DX * hxt[j]);

  u[i3D] += zderx ;

}
                                                    1,1           All
```





Real data flow graph as generated by MaxCompiler
4,866 nodes;
10,000s of stages/cycles

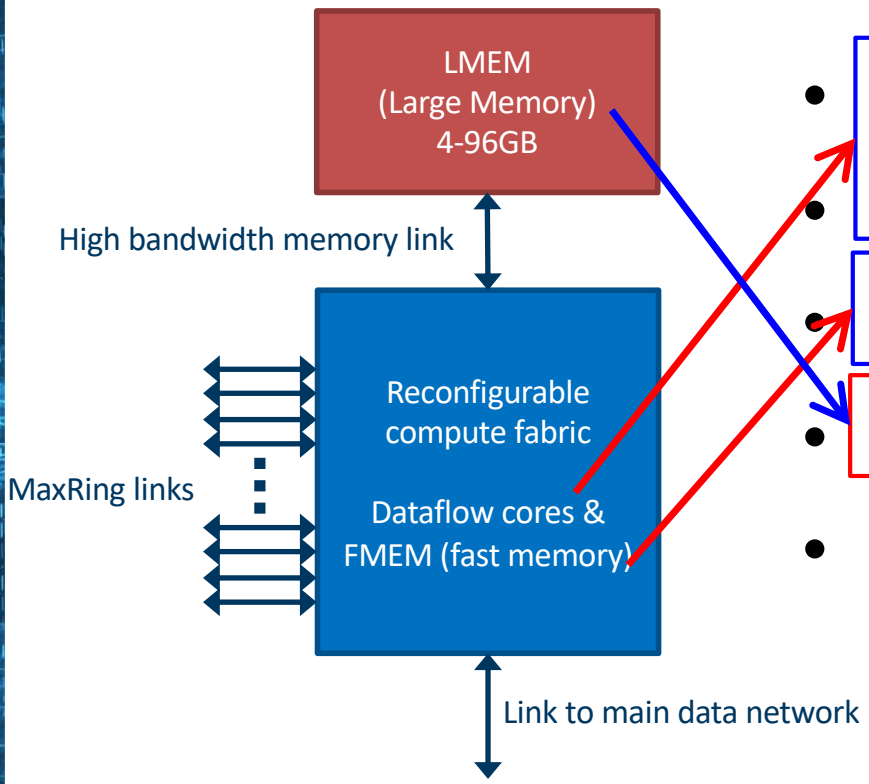
Full Customization in:
Space, Value and *Time*
(SVT)



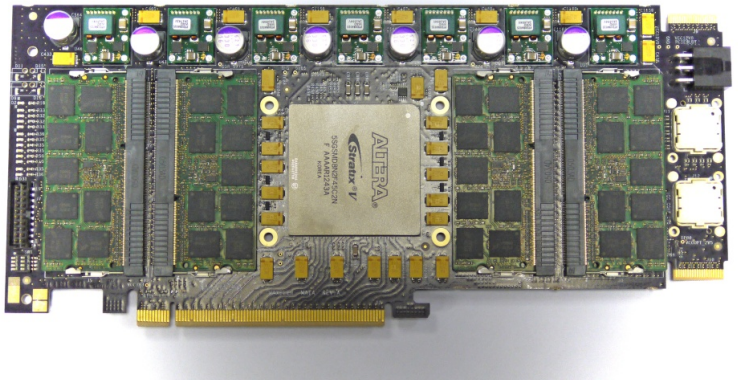
MAXELER Technologies
MAXIMUM PERFORMANCE COMPUTING

Mastering complex custom computing graphs using simple code

Maxeler's DataFlow Engines (DFEs)



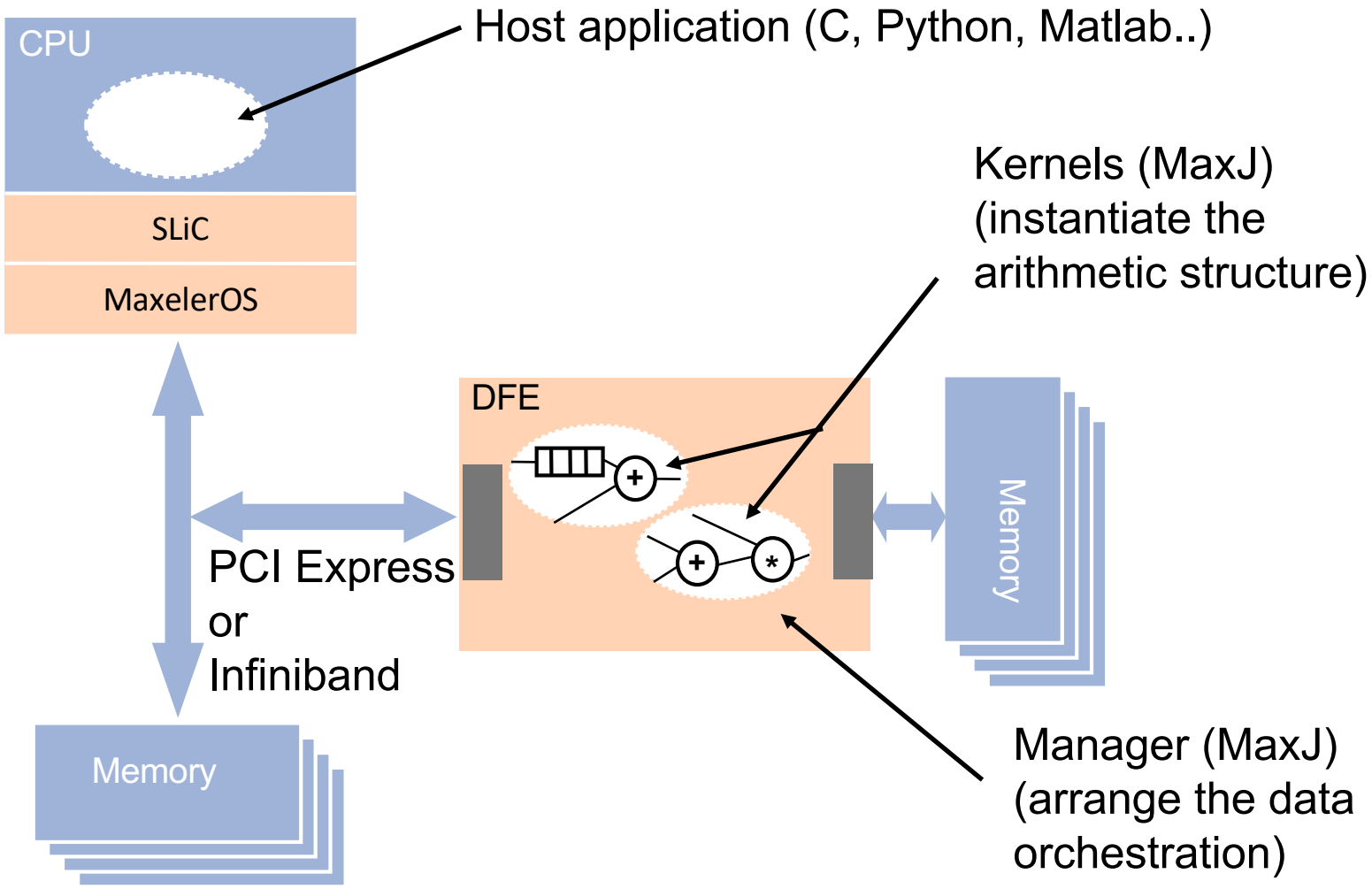
- Largest Reconfigurable Chip
- O(1k) multipliers
- O(100k) logic cells
- O(10MB) of on-chip SRAM
- O(10GB) of on-card DRAM*
- DFE-to-DFE interconnect



* working towards 128GB on a ¾, single slot PCIe card

MAXELER Technologies
MAXIMUM PERFORMANCE COMPUTING

Application Level Components



The Computational Model (DFE sub-system)

- Spatial arithmetic chip “hardware” substrate with
  - flexible arithmetic units and
  - programmable interconnect
  - (looks like FPGAs but is not limited to)
- Programmable Static Dataflow in 2D Hardware
- Systolic Execution at Kernel level
- Streaming Custom Computing at system level
- GALS* kernel-to-kernel comm.

* GALS – Globally Asynchronous Locally Synchronous

The Computational Model (SW suite)

- Complete compilation toolchain (MaxCompiler)
- Dedicated design methodology
- Integrated simulation and debug environment
- Fully integrated in Linux (CentOS)
  - runtime system (MaxelerOS and SLiC)
  - low level software support

Help designers focus on the data/algorithm and the system architecture for rapid development

[1] Nils Voss, Tobias Becker, Oskar Mencer, Georgi Gaydadjiev: *Rapid Development of Gzip with MaxJ*. ARC 2017: 60-71

The Computational Model (Memory types)

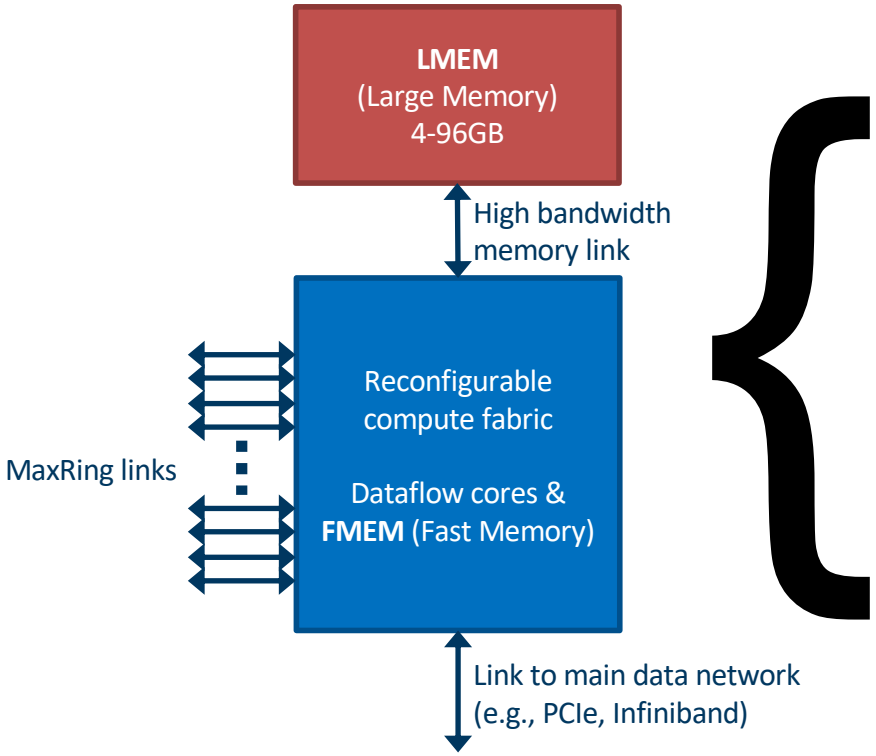
- Only three explicit basic memory types
  - Scalars (always exposed to the CPU)
  - Fast Memory (FMEM): small and fast (on-chip)
  - Large Memory (LMEM): large and slow (off-chip)
- There are, however, always exceptions, an example
  - VU9P on MAX5 has BRAM and URAM on chip
  - both constitute FMEM FPGA slices in the package
  - correct placement is a key
  - dedicated algorithms required [2]

[2] Nils Voss, Pablo Quintana, Oskar Mencer, Wayne Luk, Georgi Gaydadjiev: *Memory Mapping for Multi-die FPGAs*. FCCM 2019: pp78-86

Multiple platforms, single DFE abstraction

Application and MaxJ

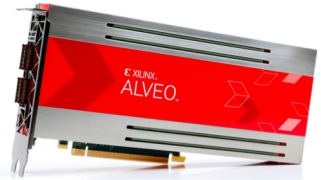
+



Performance Portable Migration

gen4

gen5



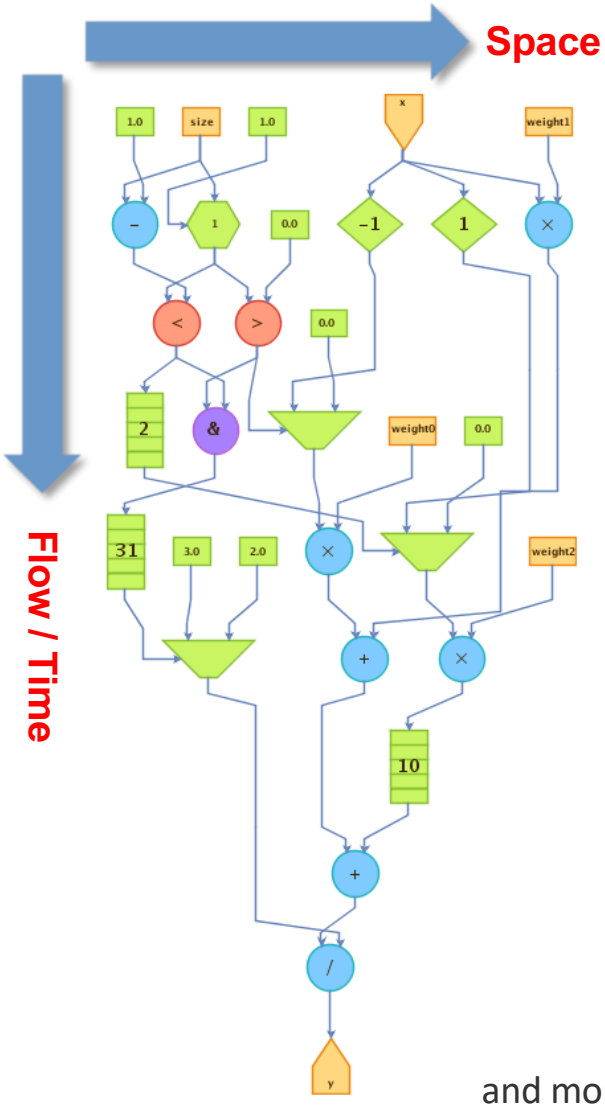
Amazon EC2 F1 Instances

(MAX4 Intel based)

(MAX5 Xilinx based)

MAXELER Technologies
MAXIMUM PERFORMANCE COMPUTING

Optimizations at all levels

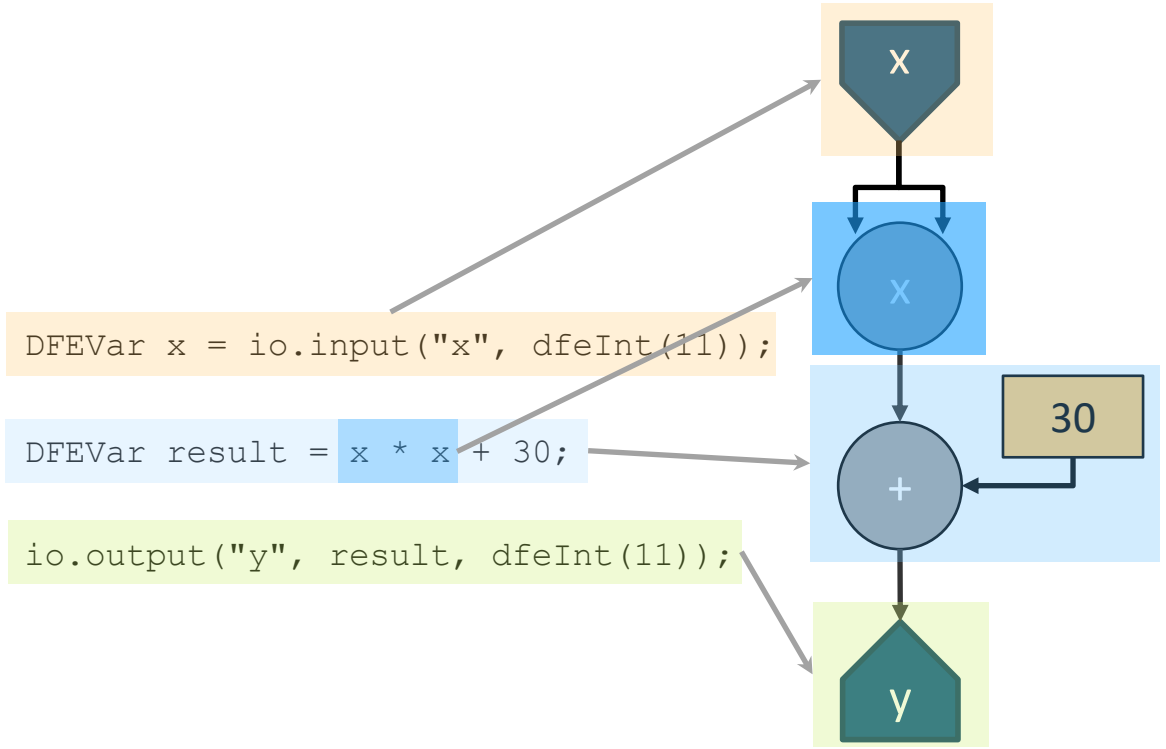


Multiple scales of computing	Important features for optimization
complete system level	⇒ balance compute, storage and IO
parallel node level	⇒ maximize utilization of compute and interconnect
microarchitecture level	⇒ minimize data movement
arithmetic level	⇒ tradeoff range, precision and accuracy =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

and more, e.g., trade/hide Communication (Time) for/behind Computation (Space)

MaxJ Basics: Programming generation of HW

- Example: $x^2 + 30$
- where x is a long 1D vector



MaxJ Basics: What is a DFEVar?

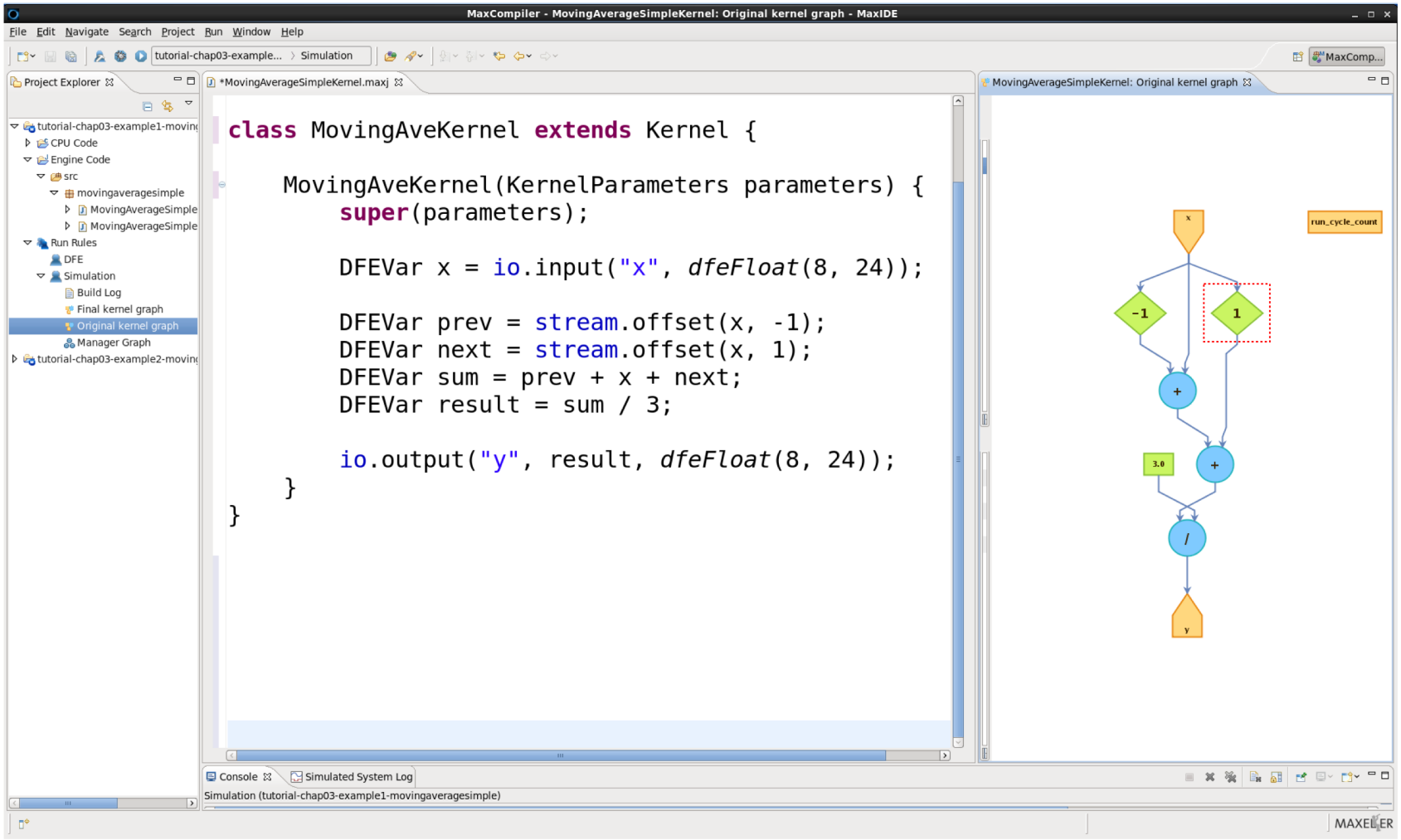
- Connection between operators in the dataflow graph
- An edge of the dataflow graph
- Stream of data elements of a certain type and size
  - (long vector)
- Physically it is a set of wires in the hardware
- It looks like a variable in MaxJ code
- BUT IS NOT A VARIABLE! (in traditional CS sense)

MaxJ Basics: Java meta-programming

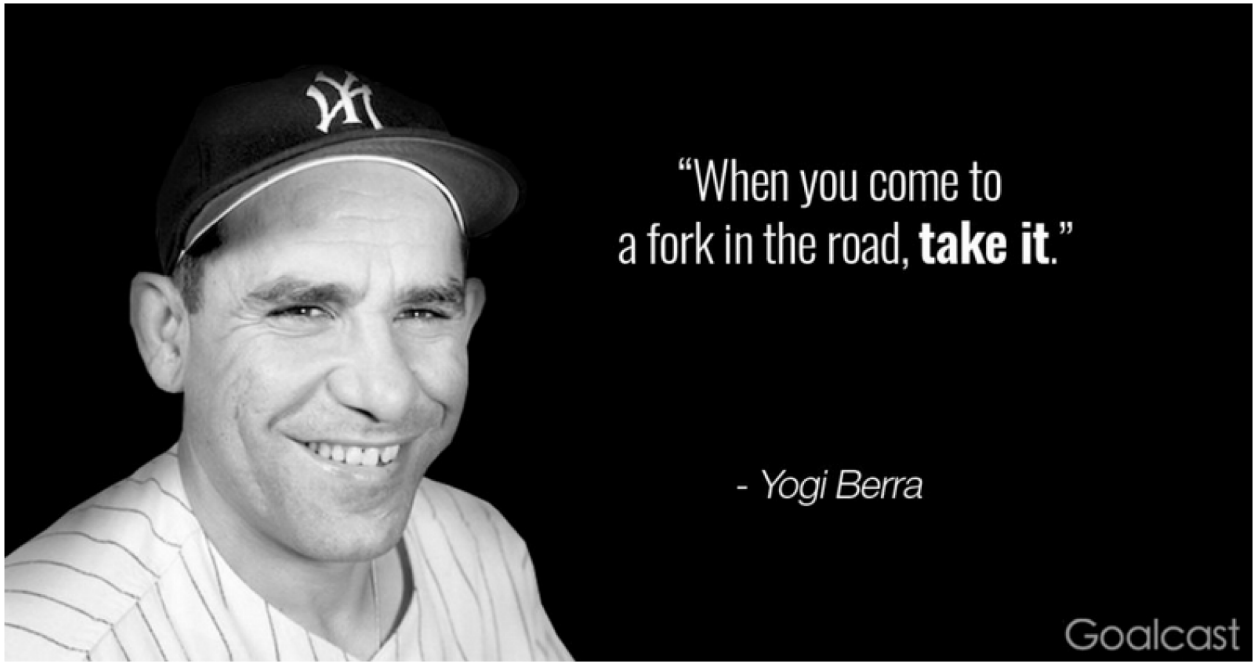
- You can use the full power of Java to write a program that *generates* the dataflow graph
- Java variables can be used as constants in hardware
  - int y; DFEVar x; x = x + y;
- Hardware variables can not be read in Java!
  - **Cannot do: int y; DFEVar x; y = x;**
- Java conditionals and loops choose how to generate hardware → not make run-time decisions
- Once you execute your Java program the generated graph is what exist in your application (not the Java)
- **We do not execute Java on the DFE!**

MaxJ: Moving Average of three numbers

Dataflow computing in hardware using a language you know

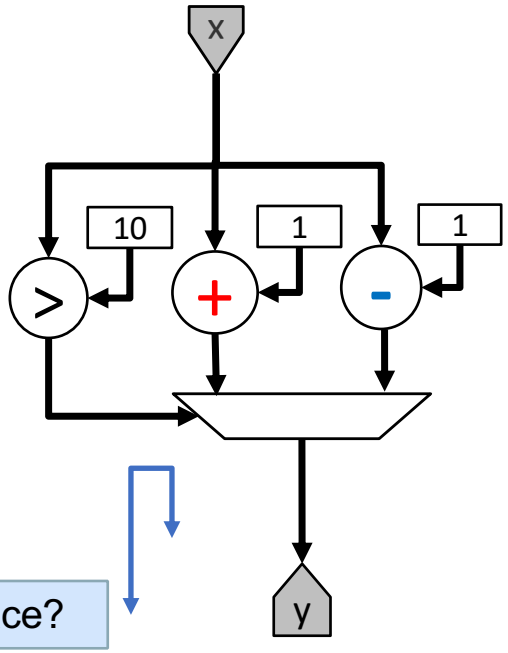


What about branches



Just turned 80 and still going strong

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



Just "take" both paths in Space?

MaxJ Basics: Java meta-programming

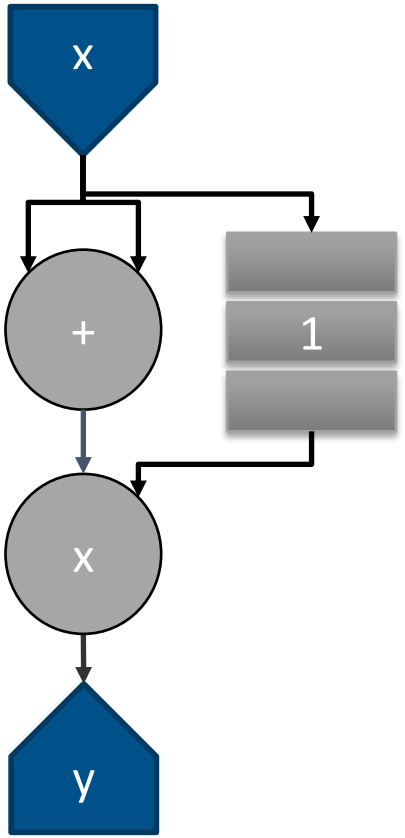
- You describe a dataflow graph generation using Java syntax
- Objects in the MaxCompiler API are used to generate hardware or configure the hardware/the build process
- Java API is crafted to ease the generation of massive dataflow graphs
  - Object Orientation possible and encouraged (e.g., using KernelLibs)
  - You can write generic code which optimises itself on the fly
  - You can write optimisation libraries, e.g., MaxPower
  - Many normal Java libraries can be used, e.g., JUnit

MaxJ Intro: Scheduling

- The dataflow graph in a kernel is statically scheduled and will be executed simultaneously in a parallel fashion
- Operations have inherent latencies
  - If different data paths meet, they need to be balanced and delays (FIFOs) are inserted
  - The scheduler tries to minimise the costs of implementing those delays in terms of FMEM
  - You can add manual scheduling constraints with `stream.offset()`

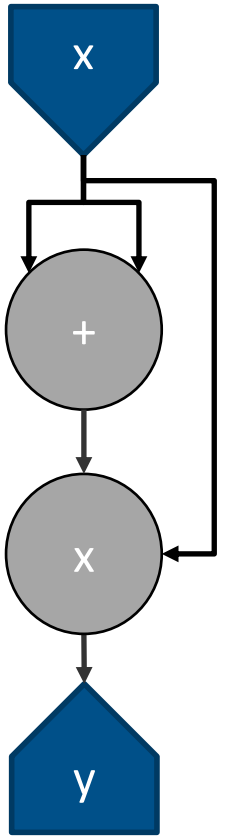
MaxJ Intro: Scheduling

```
DFEVar x = io.input("x", type);
DFEVar y;

y = (x + x) * x;

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



MaxJ Intro: Scheduling

```
DFEVar x = io.input("x", type);
DFEVar y;

y = (x + x) * stream.offset(x, 1);

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



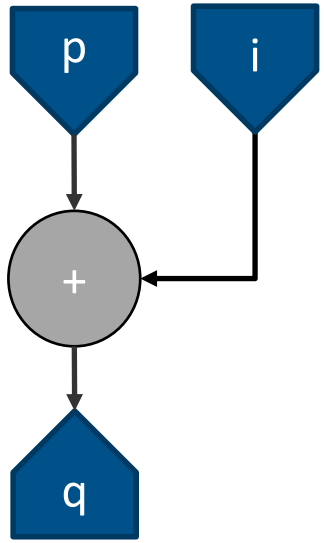
MaxJ Intro: Working with loop counters

- How can we implement this in MaxJ?

```
for (int i = 0; i < N; i++) {
          q[i] = p[i] + i;
}
```

- How about this?

```
DFEVar p = io.input("p", dfeInt(30));
DFEVar i = io.input("i", dfeInt(30));
DFEVar q = p + i;
io.output("q", q, dfeInt(31));
```



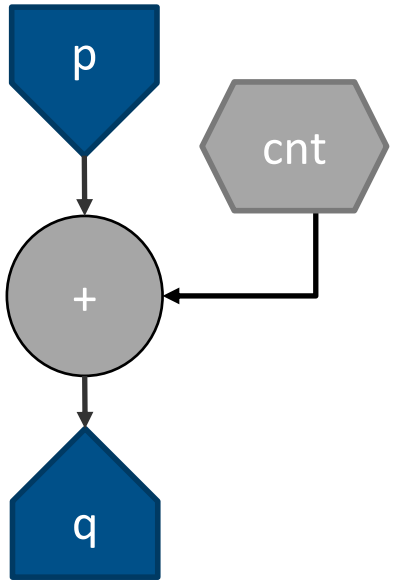
Yes.... But, now we need to create an array *i* in software and stream it to the DFE as well

MaxJ Intro: Working with loop counters

- There is very little 'information' in the *i* stream.
  - Could produce it directly on the DFE itself
  - Some HW resources will be used to do so

```
DFEVar p = io.input("p", dfeInt(31));
DFEVar i = control.count.simpleCounter(32, N);
DFEVar q = p + i;
io.output("q", q, dfeInt(32));
```

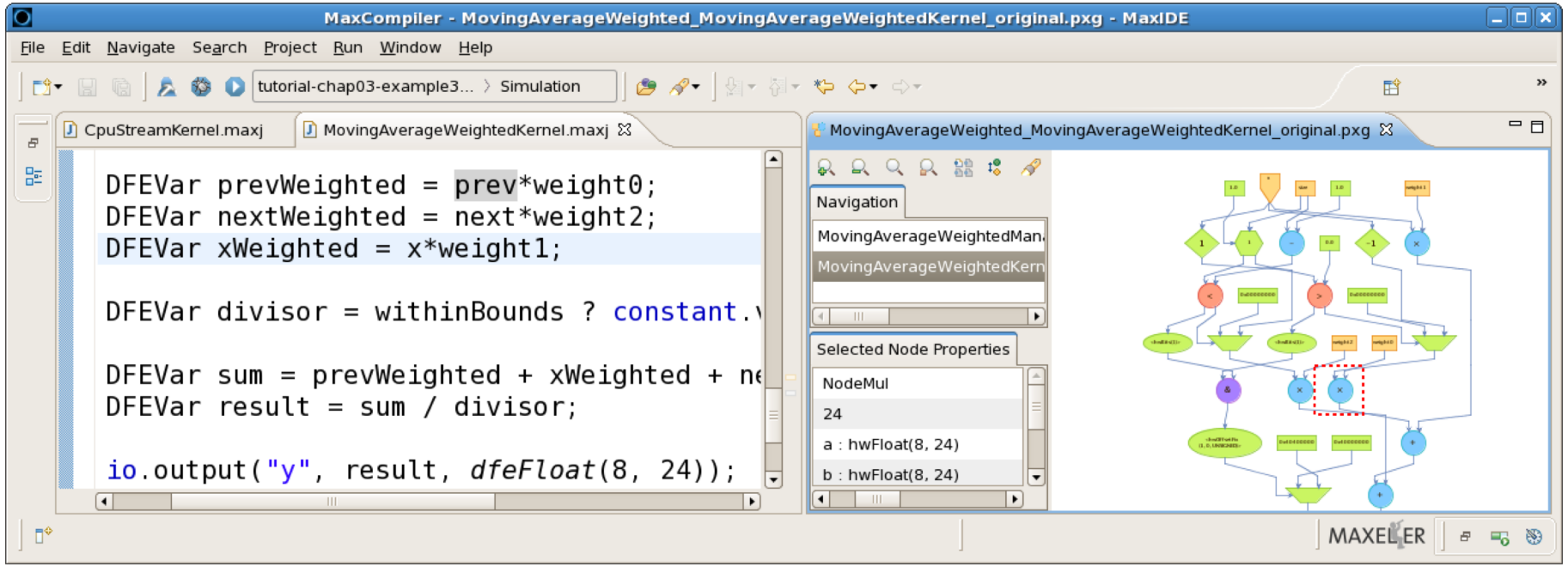
Half as many inputs
Less data transfer



- Counters can be used to generate sequences of numbers
- Complex counters can have strides, wrap points, triggers:
  - e.g., *if (y==10) y=0; else if (en==1) y=y+2;*

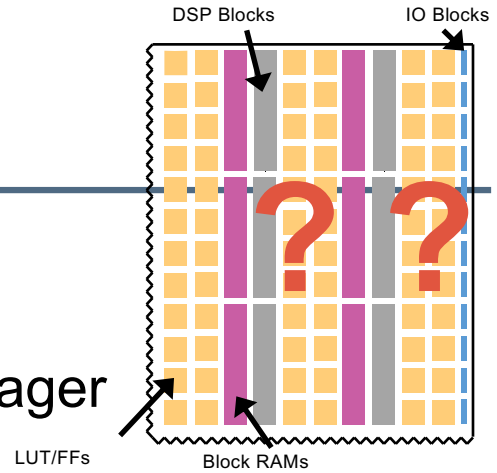
MaxJ Intro: Programming Components

- **MaxCompiler** – Java-driven dataflow compiler
- **SLiC Interface** – CPU integration
- **MaxelerOS** – optimized DFE <-> CPU link
- **Seamless simulation environment**



DFE Resource Usage Reporting

- Allows you to see what lines of code are
- using what resources and focus optimization
- Separate reports for each kernel and for the manager




```
  LUTs      FFs    BRAMs     DSPs : MyKernel.java
   727      871      1.0        2 : resources used by this file
 0.24%    0.15%    0.09%    0.10% : % of available
71.41%   61.82%  100.00%  100.00% : % of total used
94.29%   97.21%  100.00%  100.00% : % of user resources
                                  :
                                  : public class MyKernel extends Kernel {
                                  :   public MyKernel (KernelParameters parameters) {
                                  :     super(parameters);
     1       31      0.0        0 :     DFEVar p = io.input("p", dfeFloat(8,24));
     2        9      0.0        0 :     DFEVar q = io.input("q", dfeUInt(8));
                                  :     DFEVar offset = io.scalarInput("offset", dfeUInt(8));
     8        8      0.0        0 :     DFEVar addr = offset + q;
    18       40      1.0        0 :     DFEVar v = mem.romMapped("table", addr,
                                  :                              dfeFloat(8,24), 256);
   139      145      0.0        2 :     p = p * p;
   401      541      0.0        0 :     p = p + v;
                                  :     io.output("r", p, dfeFloat(8,24));
                                  :   }
                                  : }
```

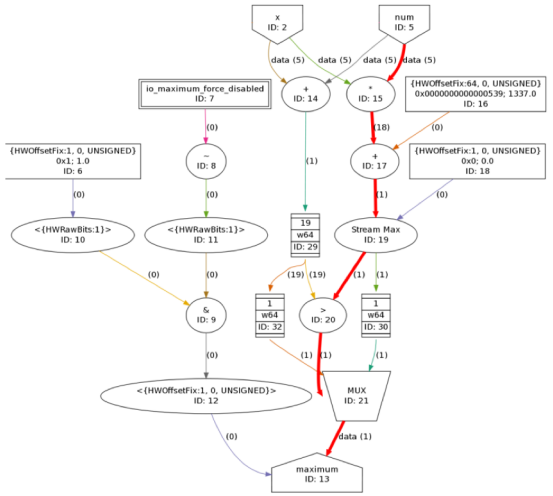
Different operations use different resources

Optimization Feedback

- MaxCompiler gives detailed latency and area annotation back to the programmer

```
27          :
28      12.8:       d.Buy = ask.Price <= lowPrice & order_book.securityId === secId;
29          :       d.Sell = bid.Price >= highPrice & order_book.securityId === secId;
30       6.4:       d.Quantity = d.Buy ? ask.Quantity : bid.Quantity;
31          :       d.Price = d.Buy ? ask.Price : bid.Price;
32
```

12.8ns + 6.4ns = 19.2ns (total compute latency)



- Evaluate precise effect of code on latency and chip area

FPGA HW vs Dataflow System Design

- DFEs currently use FPGAs
  - Maxeler MAX4C, MAX4N, MAX5C
  - Xilinx Alveo
  - Amazon EC2 F1 instance
  - (but any suitable coarser-grain technology will do)
- HPC FPGA development often focus on kernels
  - e.g., accelerate Matrix Multiply, FFT, Convolution
- I/O and memory bottlenecks are often ignored
- Dataflow looks at the complete application
  - Customise Dataflow
  - Reduce Bandwidth requirements
  - Balance the entire system
  - Maximize Throughput

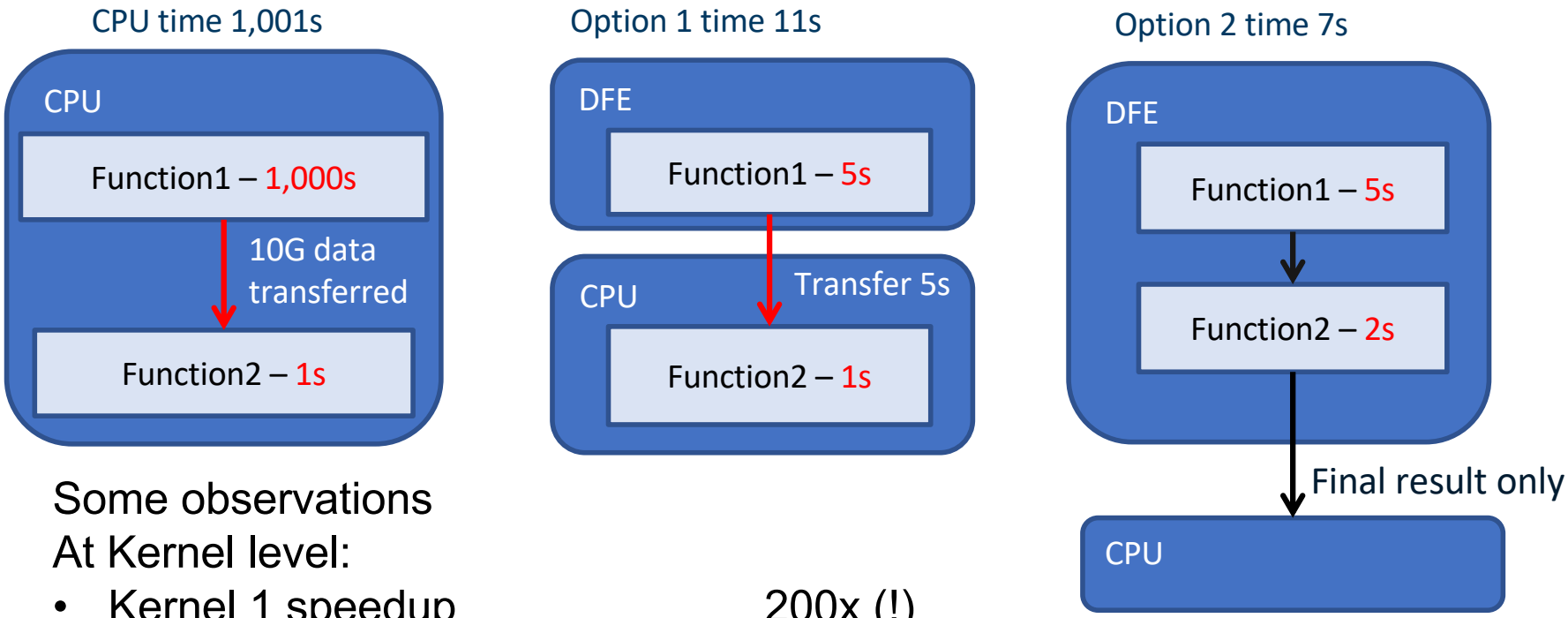
VHDL vs MaxJ example (HMS CERN)

```
class Jet extends KernelLib{
 //… constructors etc.
 public static Jet add(Jet lJet, Jet rJet) {
  DFEVar energy = lJet.energy() + rJet.energy();
  DFEVar ecal = lJet.ecal() + rJet.ecal();
  DFEVar valid = lJet.valid() & rJet.valid();
  return new Jet(lJet.kernel(), energy, ecal,
  valid);
 }
}
```

- MaxJ code is just Software
- No need to keep track on exact bit sizes, fixed point positions, etc
- Other kernels benefit even more
  - Bitonic Sort is 500 VHDL lines versus 130 in MaxJ

```
ARCHITECTURE behavioral OF JetSum IS
  SIGNAL jetTmp : tJet := cEmptyJet;
  SIGNAL EnergyTmp , EcalTmp :
STD_LOGIC_VECTOR( 21 DOWNTO 0 );
BEGIN
  PROCESS( clk )
  BEGIN
    IF( RISING_EDGE( clk ) ) THEN
      IF( NOT jetIn1.DataValid ) THEN
        jetOut <= cEmptyJet;
      ELSE
        jetOut.Energy( 15 DOWNTO 0 ) <=
            jetIn1.Energy + jetIn2.Energy;
        jetOut.Ecal( 15 DOWNTO 0 )   <=
             jetIn1.Ecal + jetIn2.Ecal;
        jetOut.DataValid <= TRUE;
      END IF;
    END IF;
  END PROCESS;
END ARCHITECTURE behavioral;
```

System Example: Decelerate to Accelerate



Some observations

At Kernel level:

- Kernel 1 speedup 200x (!)
- Kernel 2 “speedup” 0.5x (!)

At System level:

- Option 1 (Kernel 1 only) speedup 91x
- Option 2 (Kernels 1 and 2) speedup 143x

But what about the required effort?

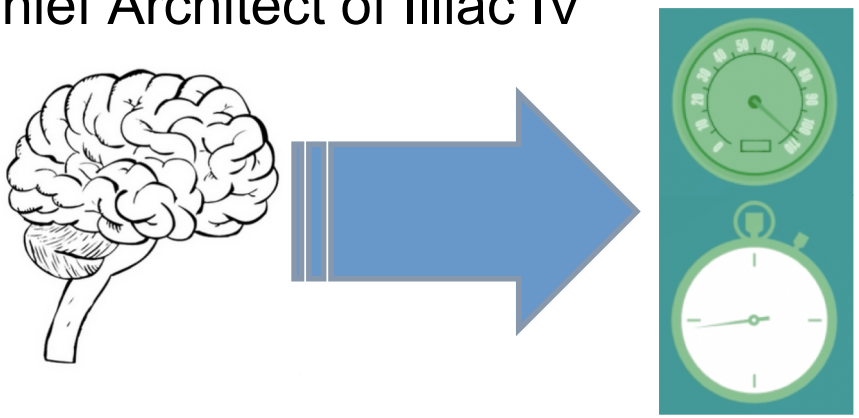
Easy it is not (and not really new)

Slotnick's law (of effort):

"The parallel approach to computing does require that some ***original thinking*** be done **about numerical analysis and data management** in order to secure efficient use.

In an environment which has represented the absence of the need to think as the highest virtue this is a decided disadvantage."

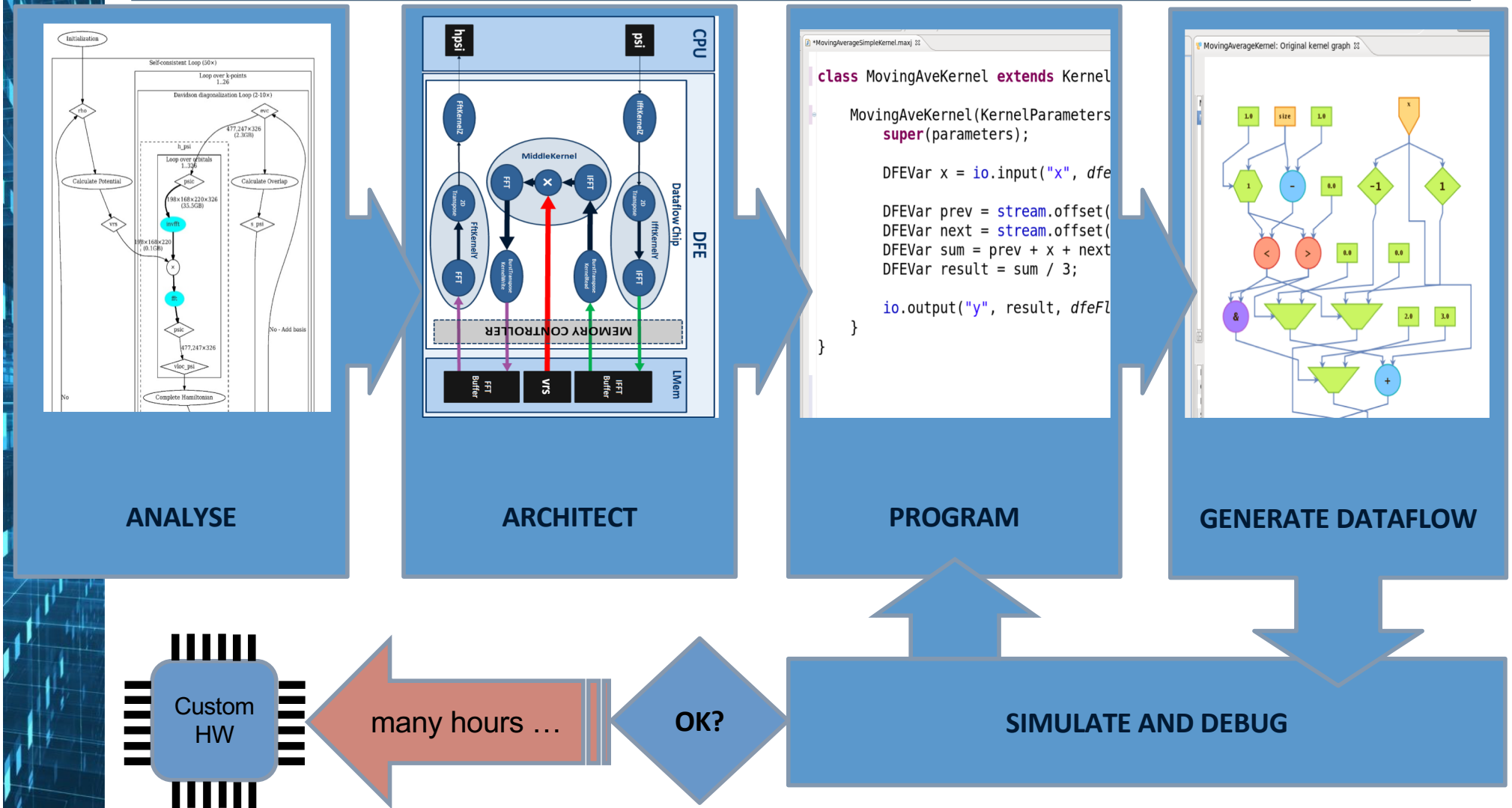
Daniel Slotnick (1931-1985)
Chief Architect of Illiac IV



Turn brain power into performance

MAXELER Technologies
MAXIMUM PERFORMANCE COMPUTING

Non Traditional Design Process

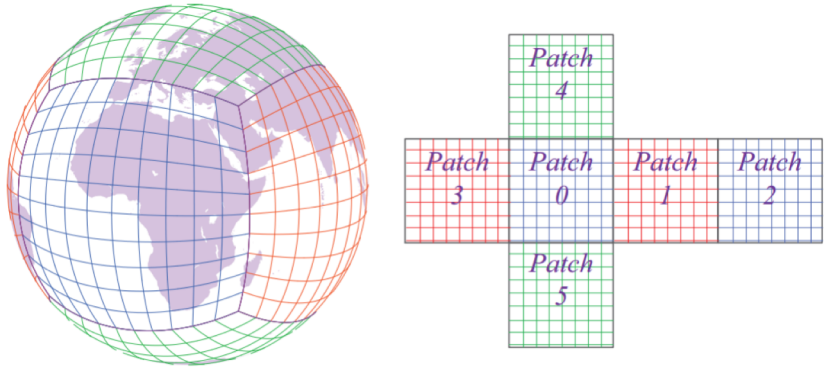


*Used to build **balanced** real systems, however, not easy to learn/educate*

Global Weather Simulation with DFEs in China

An order of magnitude improvement over the Linpack-driven supercomputer technology

- L. Gan, H. Fu, W. Luk, C. Yang, W. Xue, X. Huang, Y. Zhang, and G. Yang, *Accelerating solvers for global atmospheric equations through mixed-precision data flow engine*, published at **FPL 2013**
- Joint research with Imperial College and Tsinghua University
- Simulating the atmosphere using the shallow water equation



(a) The cubed-sphere mesh (b) The computational domain

Fig. 1. Mesh and computational domain.

Platform	Speedup	Energy Efficiency
6 Core CPU	**1x**	**1x**
Tianhe-1A Node	**23x**	**15x**
Maxeler MPC-X	**330x**	**145x**

Imperial College London

清華大學 Tsinghua University

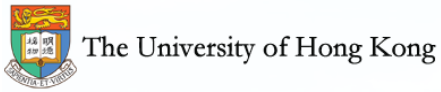
Open Research Questions

- How to compare against general purpose machines?
- How to validate results as “good enough”?
- How to forecast the required “Dan Slotnick’s Effort”?
- How to significantly decrease P&R times?
- How to create a much better silicon substrate?
- How to educate the *think* ➔ *model* ➔ *program* mindset?
- Tools, tools, tools, …

(we need the help of Parallel Programming experts, you)

All of the above while dealing with the growing impact of Quantum Mechanical Effects on running software

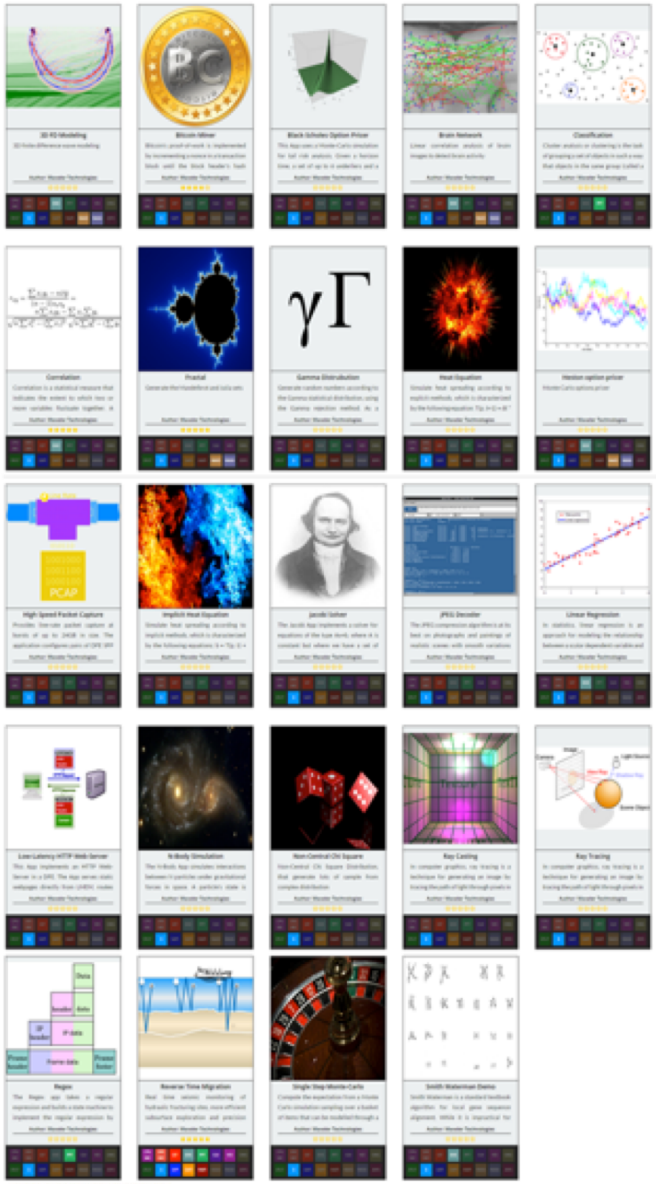
Over 150 Maxeler University Program Members



MAXELER
Technologies
MAXIMUM PERFORMANCE COMPUTING

Maxeler Applications Gallery

Dataflow Apps and Analytics for Machine Learning http://appgallery.maxeler.com/



Dataflow Engine (DFE) Ecosystem

- With over 150 universities in our university program, we decided to create an app gallery to enable the community to share applications, examples, demos, …
- The App Gallery is complemented by a teaching program, with the first successful course taught at Imperial College in 2014. see http://cc.doc.ic.ac.uk/openspl16
- Top 10 APPS:
  - ➢ Correlation: in real-time, pairwise, on 6,000 streams
  - ➢ 100% Guaranteed Packet Capture
  - ➢ Webserver, cache and load balancing
  - ➢ HESTON Option pricer
  - ➢ N-body simulation
  - ➢ Regex matching (e.g. for Security)
  - ➢ Brain network simulation
  - ➢ Quantum Chromo-Dynamics kernel
  - ➢ Seismic Imaging
  - ➢ Realtime Classification

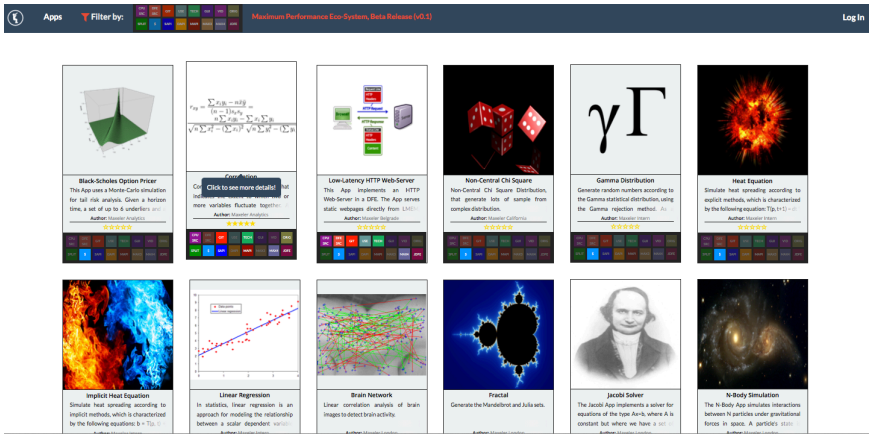
Some links with more information

Maxeler Multiscale Dataflow Computing:
https://www.maxeler.com/technology/dataflow-computing/

Computing in Space explained by Mike Flynn:
http://www.openspl.org/what-is-openspl/

Computing in Space Course at Imperial College:
http://cc.doc.ic.ac.uk/openspl16/

Exciting Applications for DFEs (and JDFEs):
http://appgallery.maxeler.com



Maxeler DFEs on AWS EC2 F1:
https://aws.amazon.com/marketplace/seller-profile?id=2780c6ec-d326-47fc-9ff6-c66ab2ba202a

Maxeler and Xilinx Alveo collaboration:
https://www.xilinx.com/products/boards-and-kits/alveo.html

georgi@maxeler.com

Questions