

High-level programming of data-centric reconfigurable dataflow systems

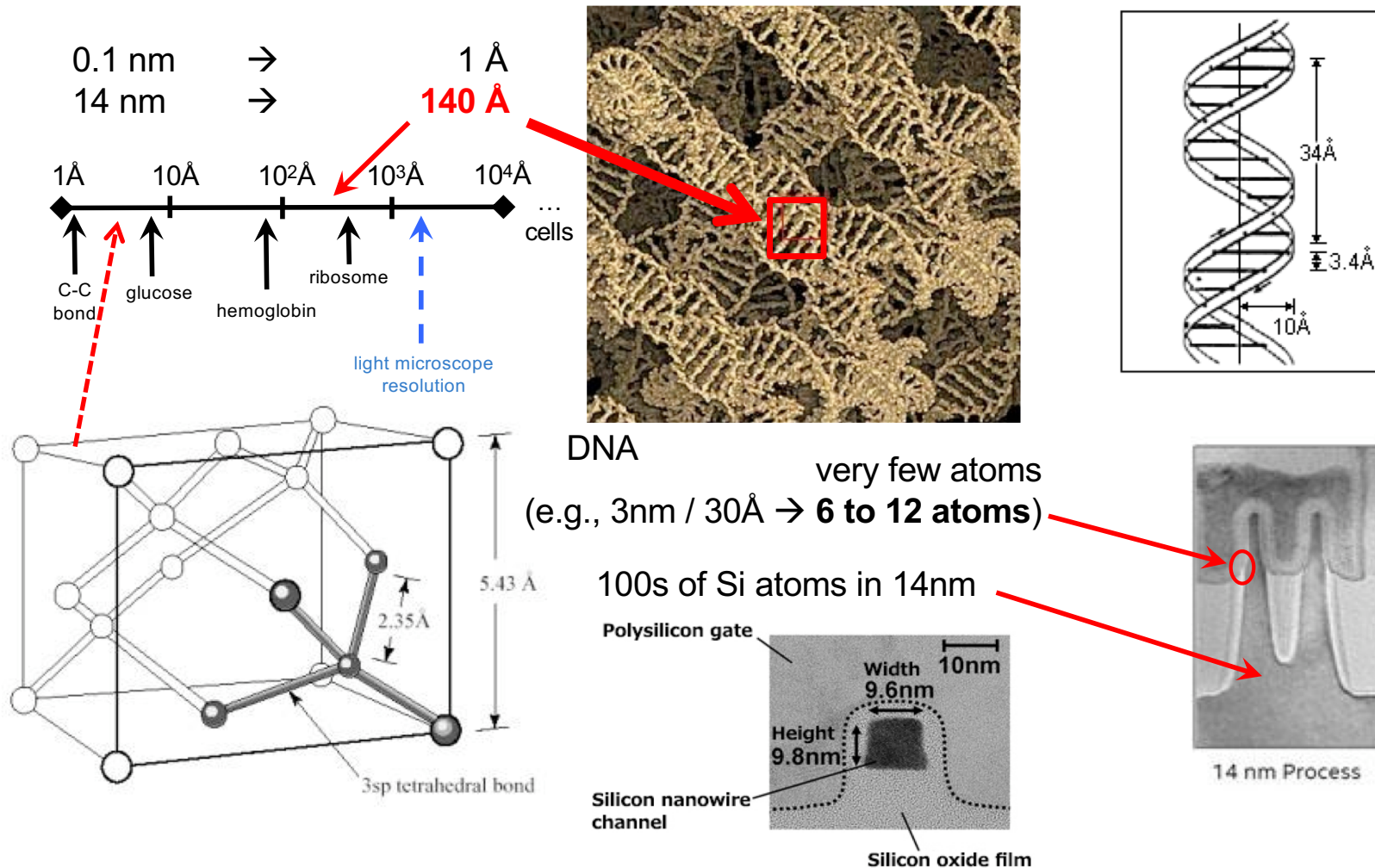
Georgi Gaydadjiev, Director of Maxeler IoT-Labs BV, Delft
Honorary Visiting Professor at the Department of Computing, Imperial College London



*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



The Data Movement Challenge

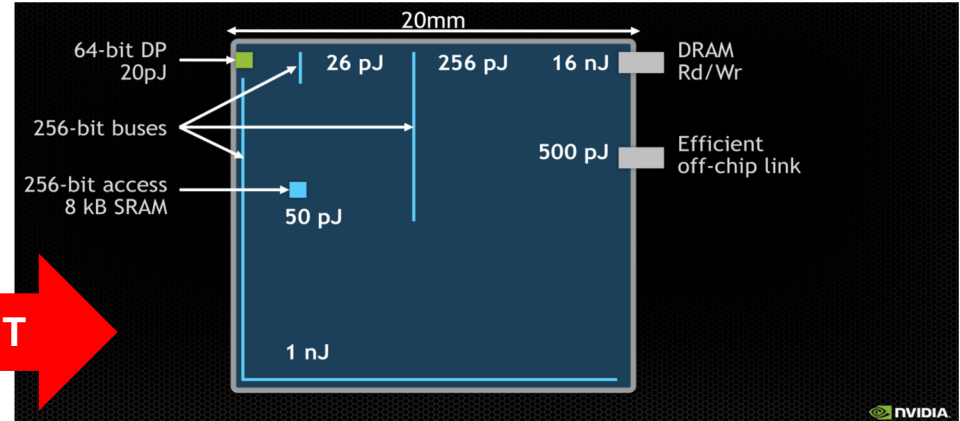
DFMA 0.01mm² 10pJ/OP – 2GFLOPs

A chip with 10⁴ FPU's:
100mm²
200W
20TFLOPS

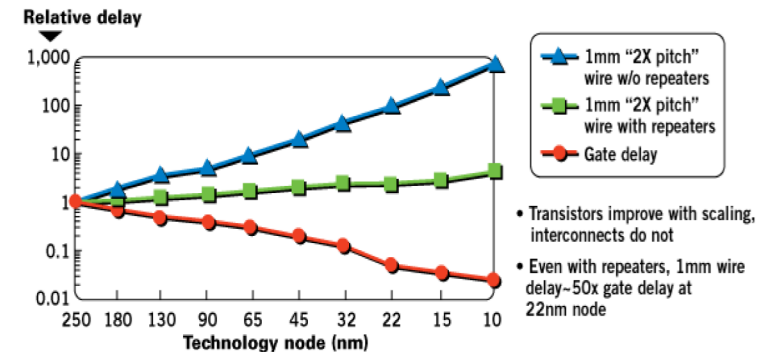
Pack 50,000 of these in racks
1EFLOPS
10MW

16nm chip, 10mm on a side, 200W

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



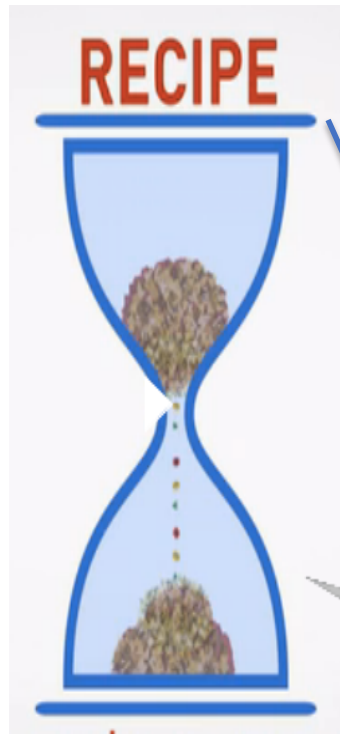
Source: ITRS

(Courtesy: NVIDIA and ITRS)

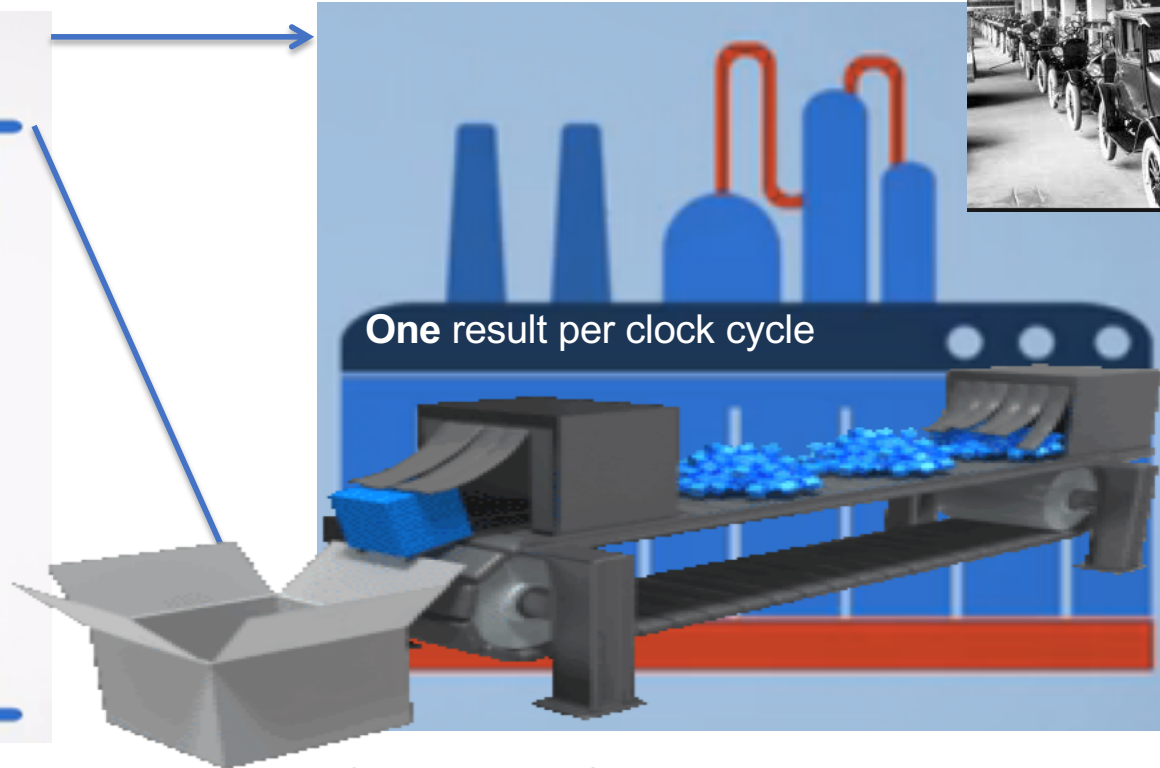
Moving data off-chip will cost ~200x more energy and is also much slower

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*

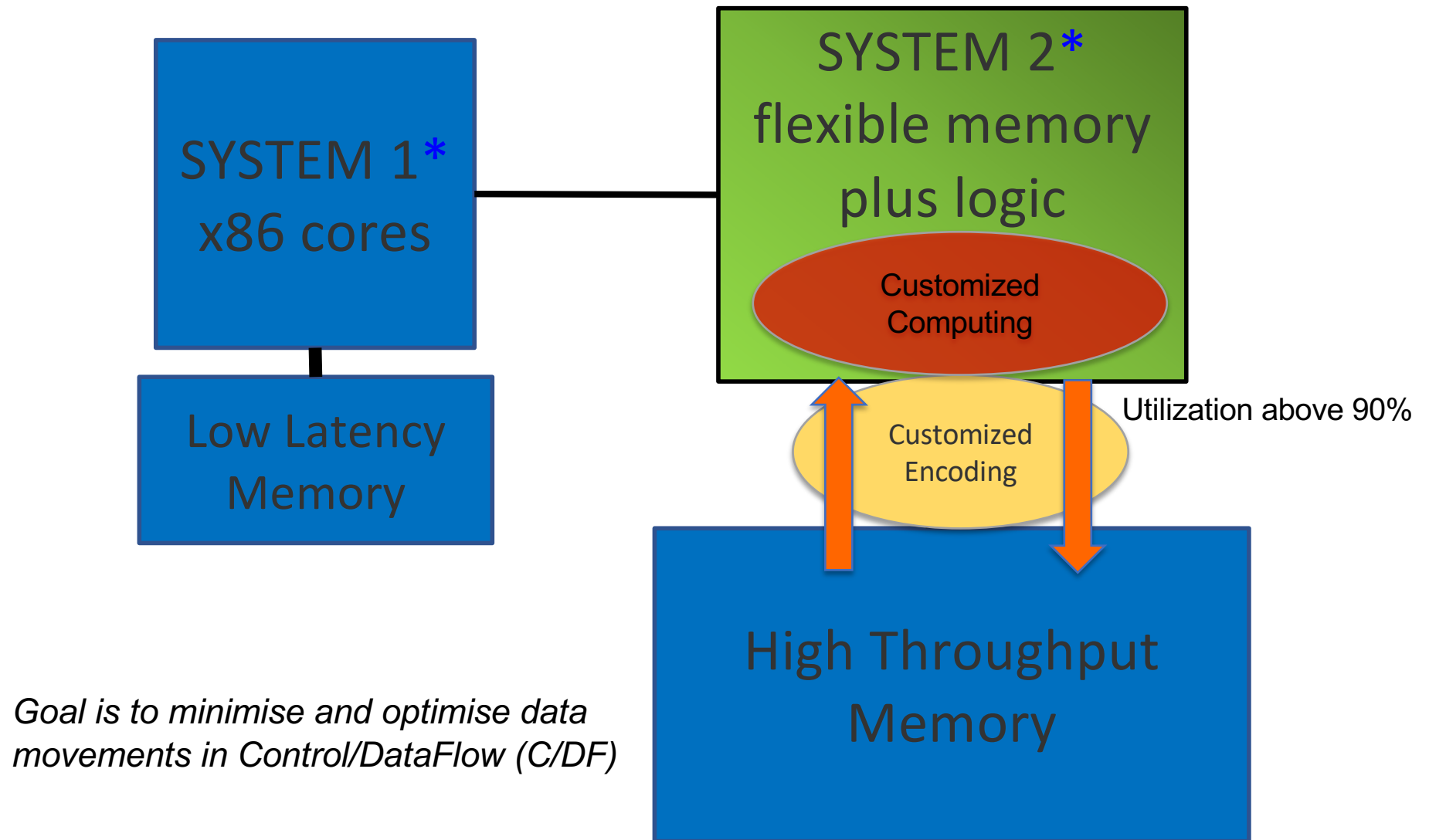


Computing in Space:
*Build a "recipe specific" factory with multiple
paths performed simultaneously*

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

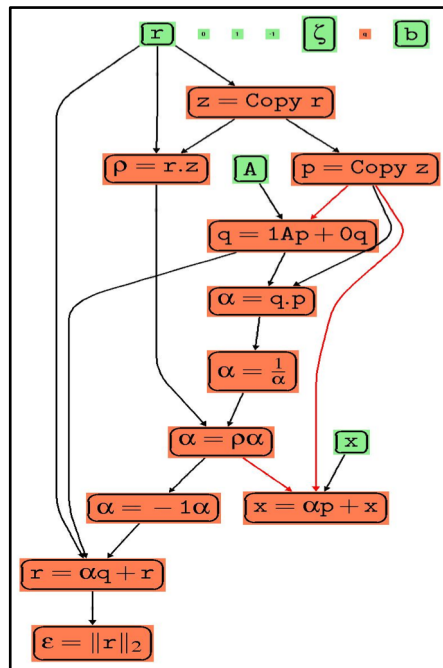


* System 1 and System 2 are based on D Kahneman,
"Thinking Fast Thinking Slow", Nobel Prize in Economics, 2002

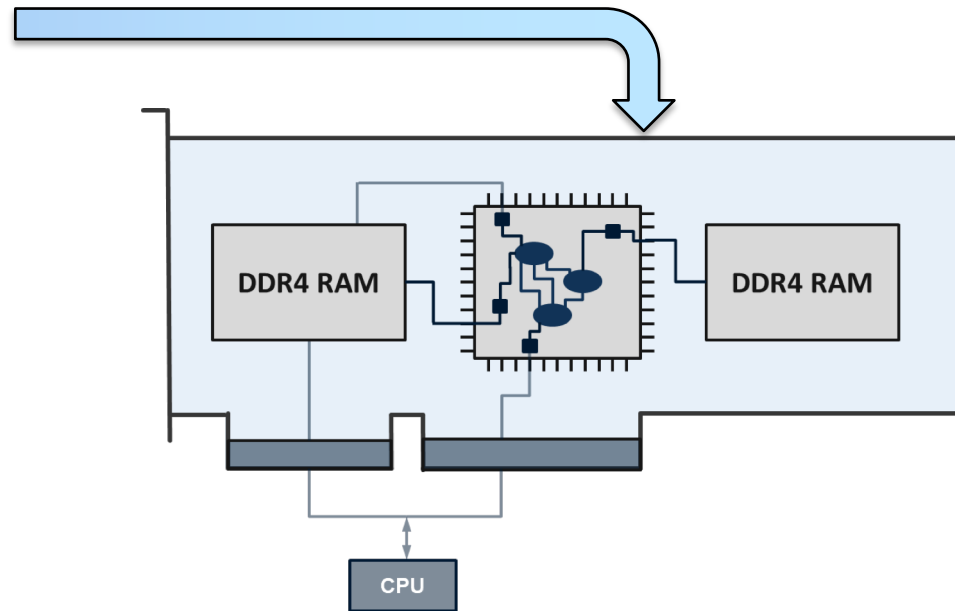
Programming a Dataflow “mass production” Engine

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

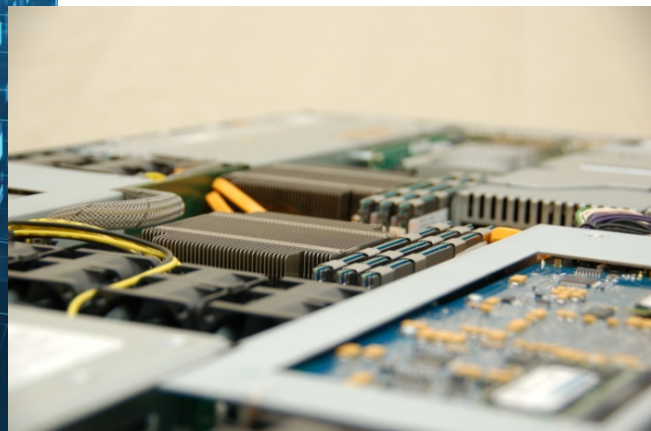
```
Vi Editor: meteo.c
File Edit View Terminal Tabs Help
/* Compute Virtual Temperature contribution*/
for(int i3D=0; i3D<NLAT*NLON*NLEV; i3D++) {

    float alps = log(ps[i2D]);
    float alps_pl = log(ps[i2D + 1]);
    float zdxlps = RD * 0.5f * (alps_pl - alps);
    float exrcp = exp(RCP * alps_pl);
    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 ;

}
```

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



```
er/clients/georgetown/meteo/MeteoKernelOpt.java - MaxIDE
Run Window Help
derivative of virtual temperature
com.maxeler.c
import declar
MeteoKernelOp
RD_C : dout
FV_C : doub
ALFK_C : do
BETK_C : do
PHIG_C : dou
INV_DX_C :
PSBCM_C : c
PSP_C : dou
TBCM_C : dc
U_C : double
Q_C : double
UBIG_C : dou

DFEVar alps = KernelMath.log(inputRange, ps_mod, floatType);
DFEVar zdxlps_mod = RD * 0.5 * (stream.offset(alps, 1) - alps);
PowOfConstant thePow = new PowOfConstant(RCP);
DFEVar exrcp = thePow.funcEval(ps, floatType);
DFEVar tvirt = (1 + EP * q) * t * exrcp * SIGRCP;
DFEVar zderx = - (tvirt + stream.offset(tvirt, NLAT))
                * zdxlps * INV_DX * INV_HXT;

DFEVar up = zderx;
```

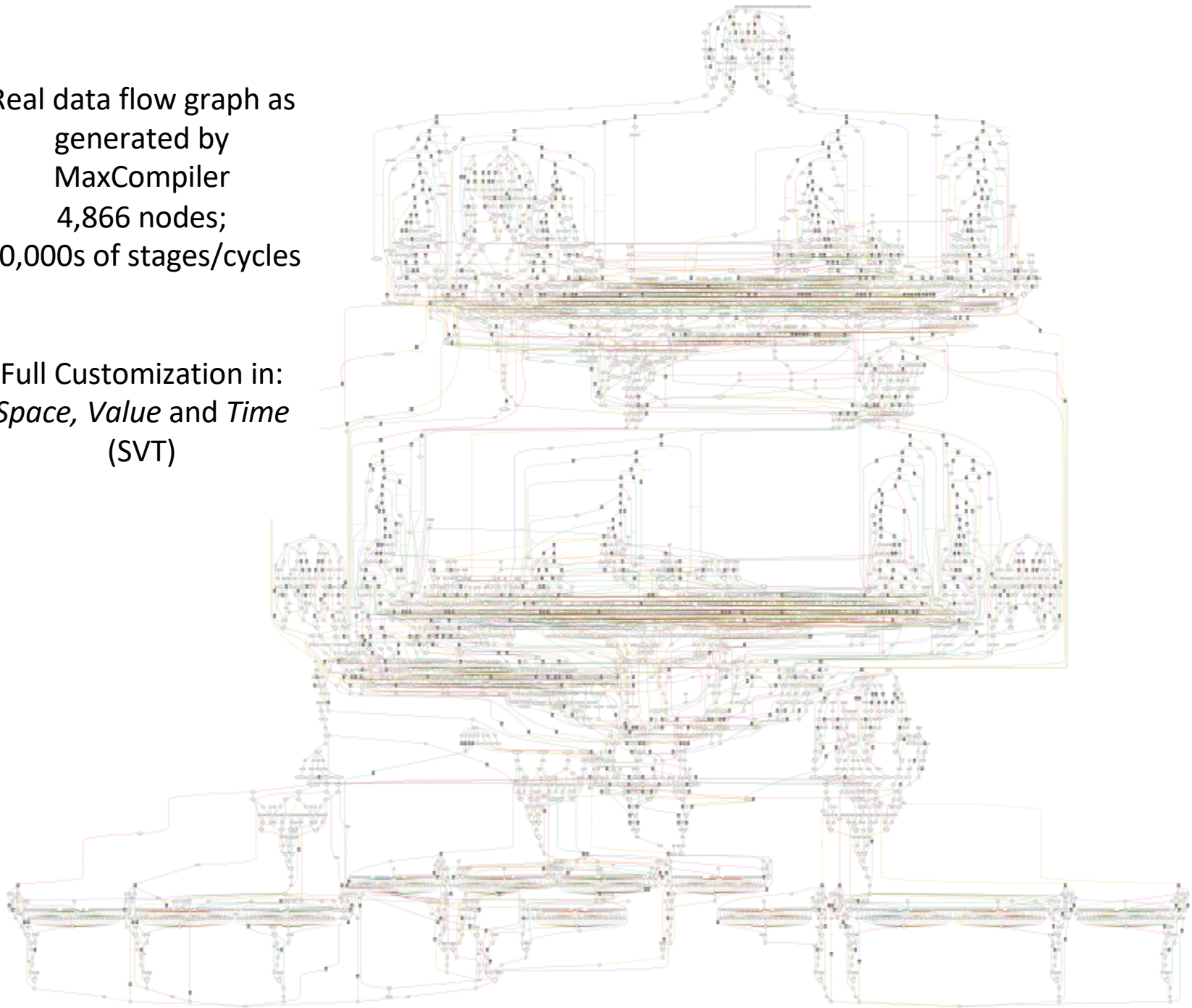
Console @ javadoc

```
com.maxeler.clients.georgetown.meteo.lib.PowOfConstant
Implementation of the power of constant function.
Author:
```

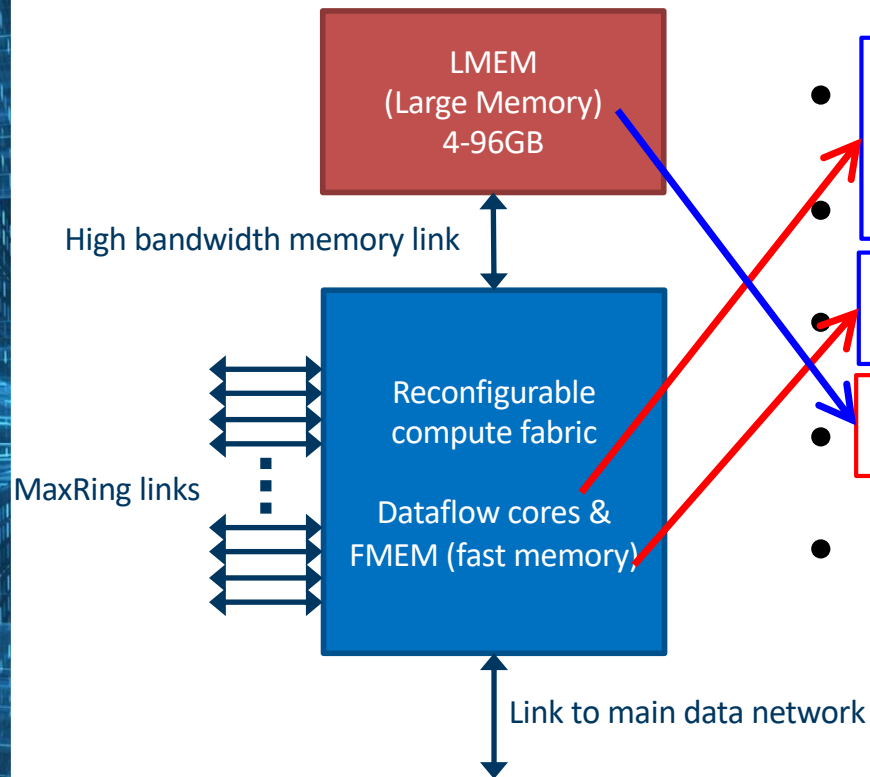
Writable Smart Insert 433 : 15

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

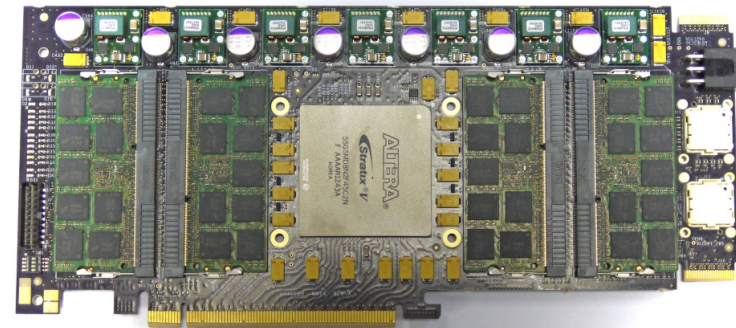
Full Customization in:
Space, Value and Time
(SVT)



Maxeler's DataFlow Engines (DFEs)

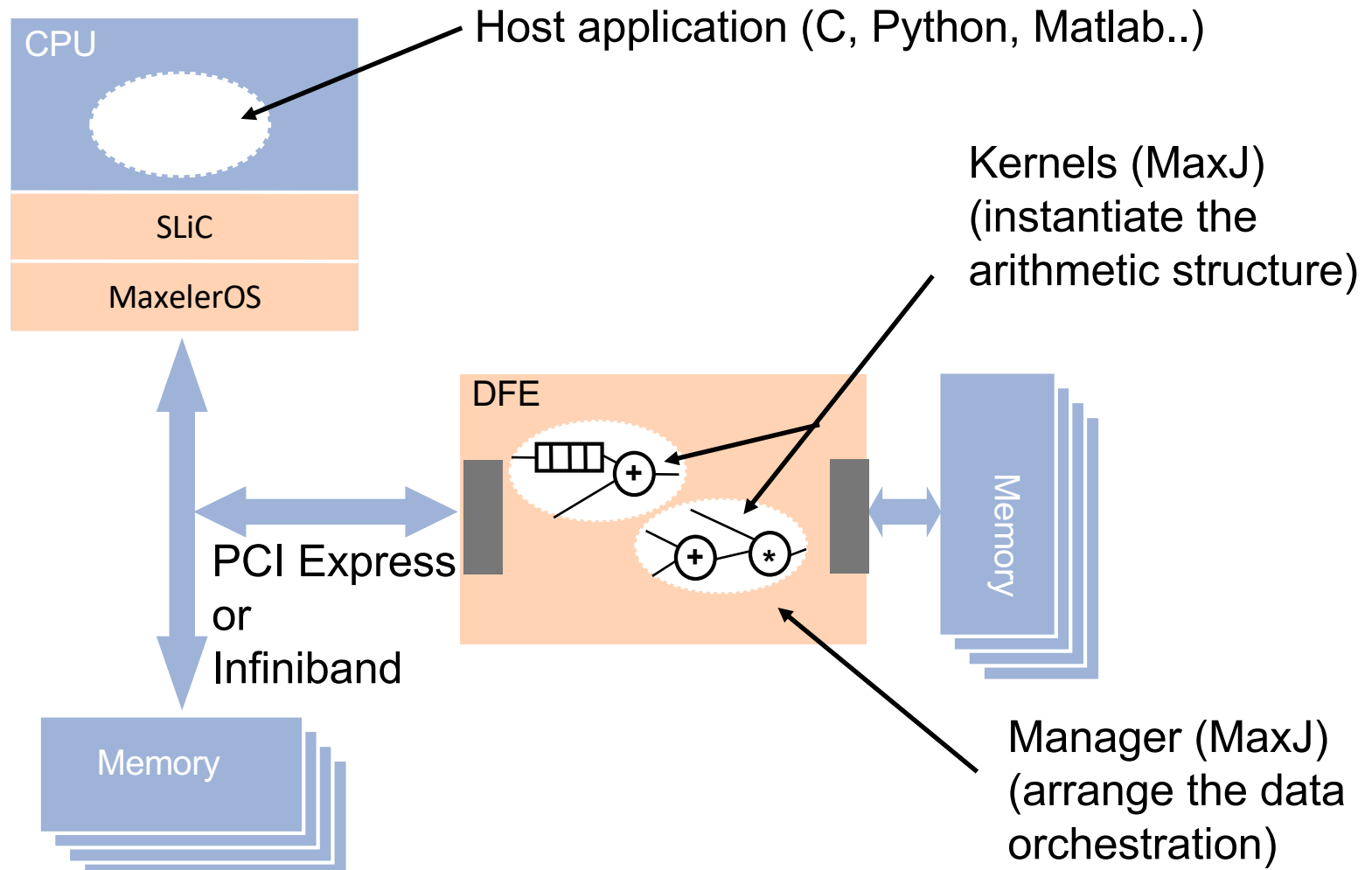


- Largest Reconfigurable Chip
- $O(1k)$ multipliers
- $O(100k)$ logic cells
- $O(10MB)$ of on-chip SRAM
- $O(10GB)$ of on-card DRAM*
- DFE-to-DFE interconnect



* working towards 128GB on a $\frac{3}{4}$, single slot PCIe card

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

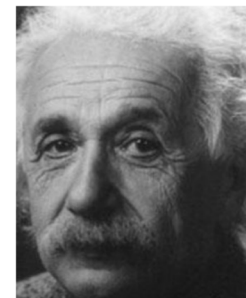


The Computational Model (SW suite)

- 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



Make everything as simple as possible, but not simpler.

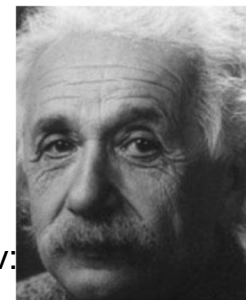
— Albert Einstein —

AZ QUOTES

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



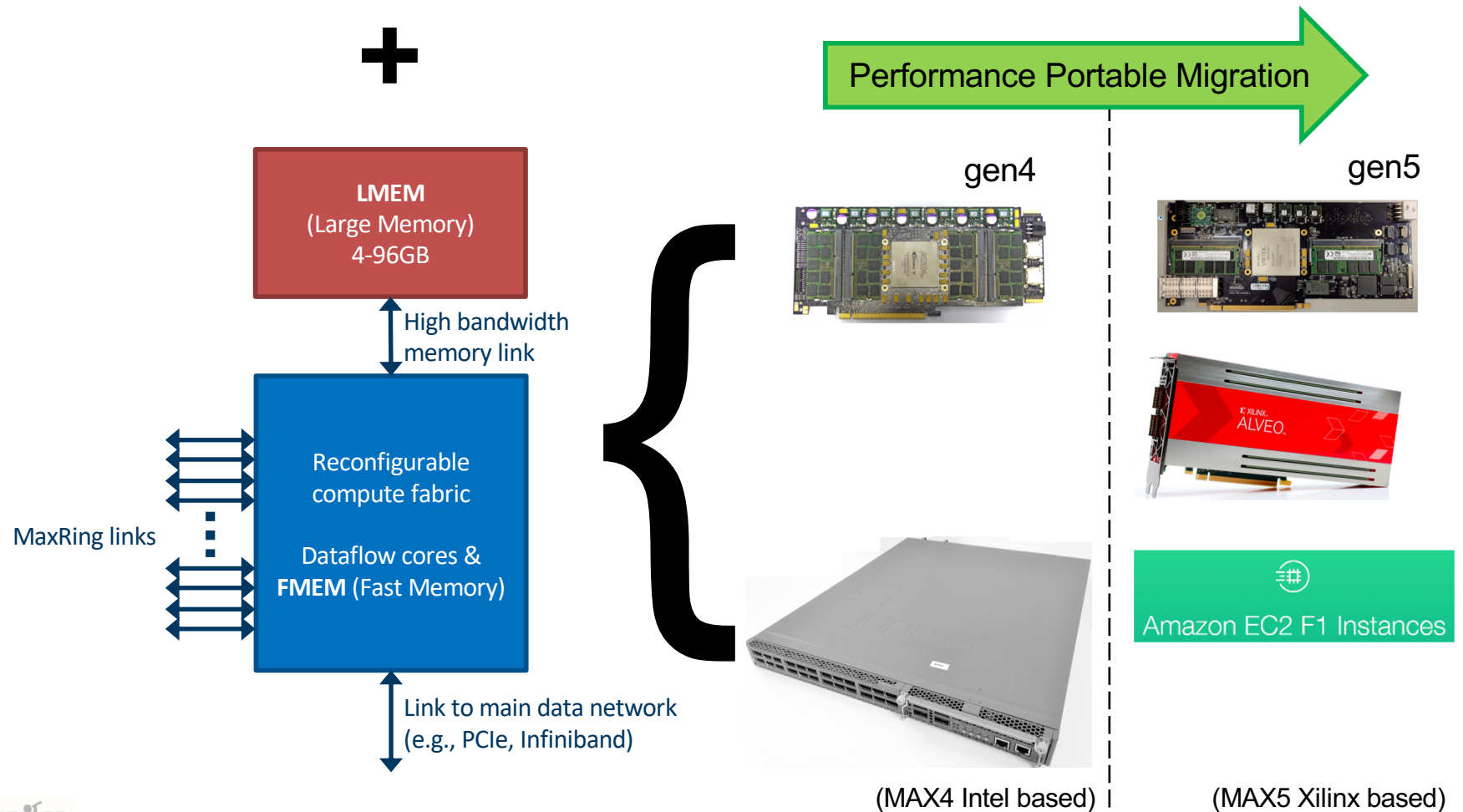
Make everything as simple as possible, but not simpler.

— Albert Einstein —

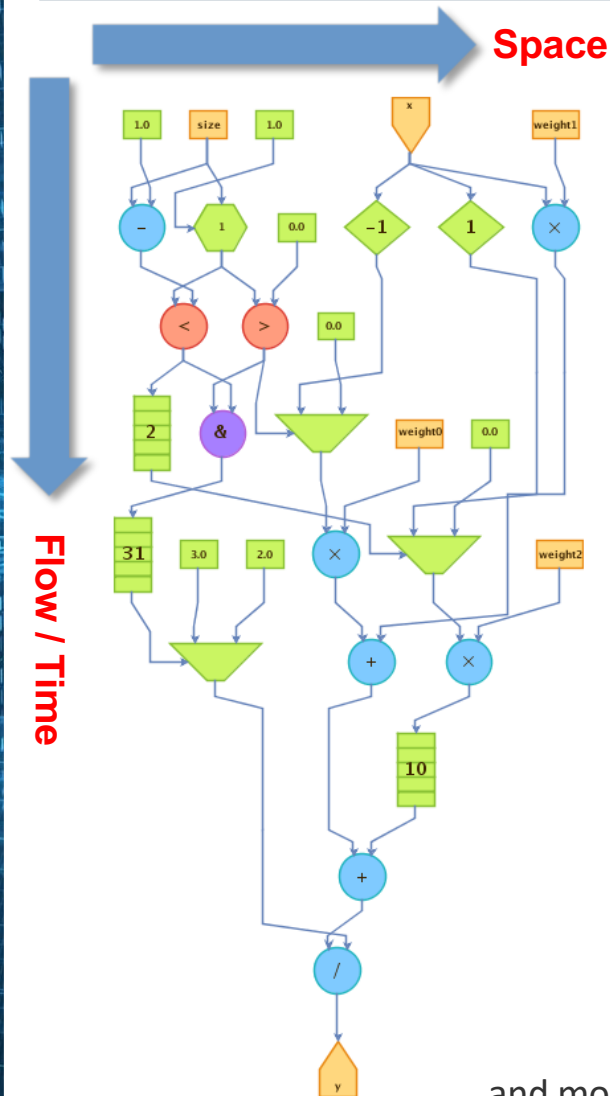
AZ QUOTES

Multiple platforms, single DFE abstraction

Application and MaxJ



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

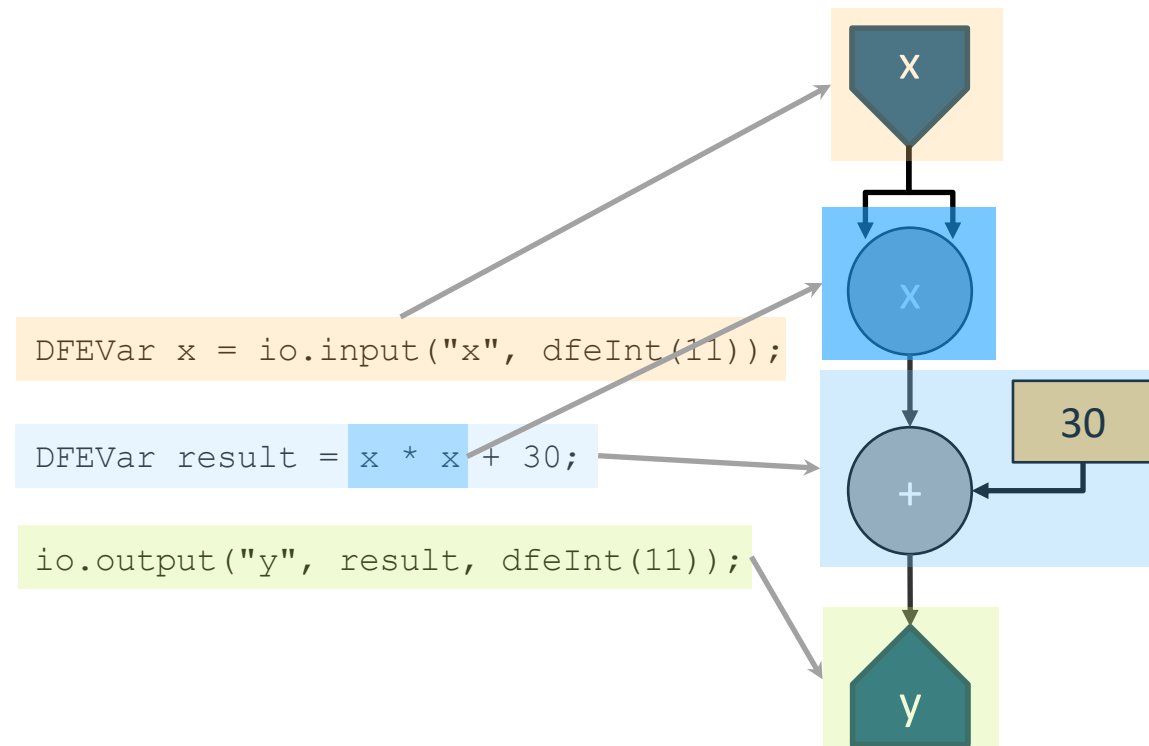
transistor level

⇒ 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?

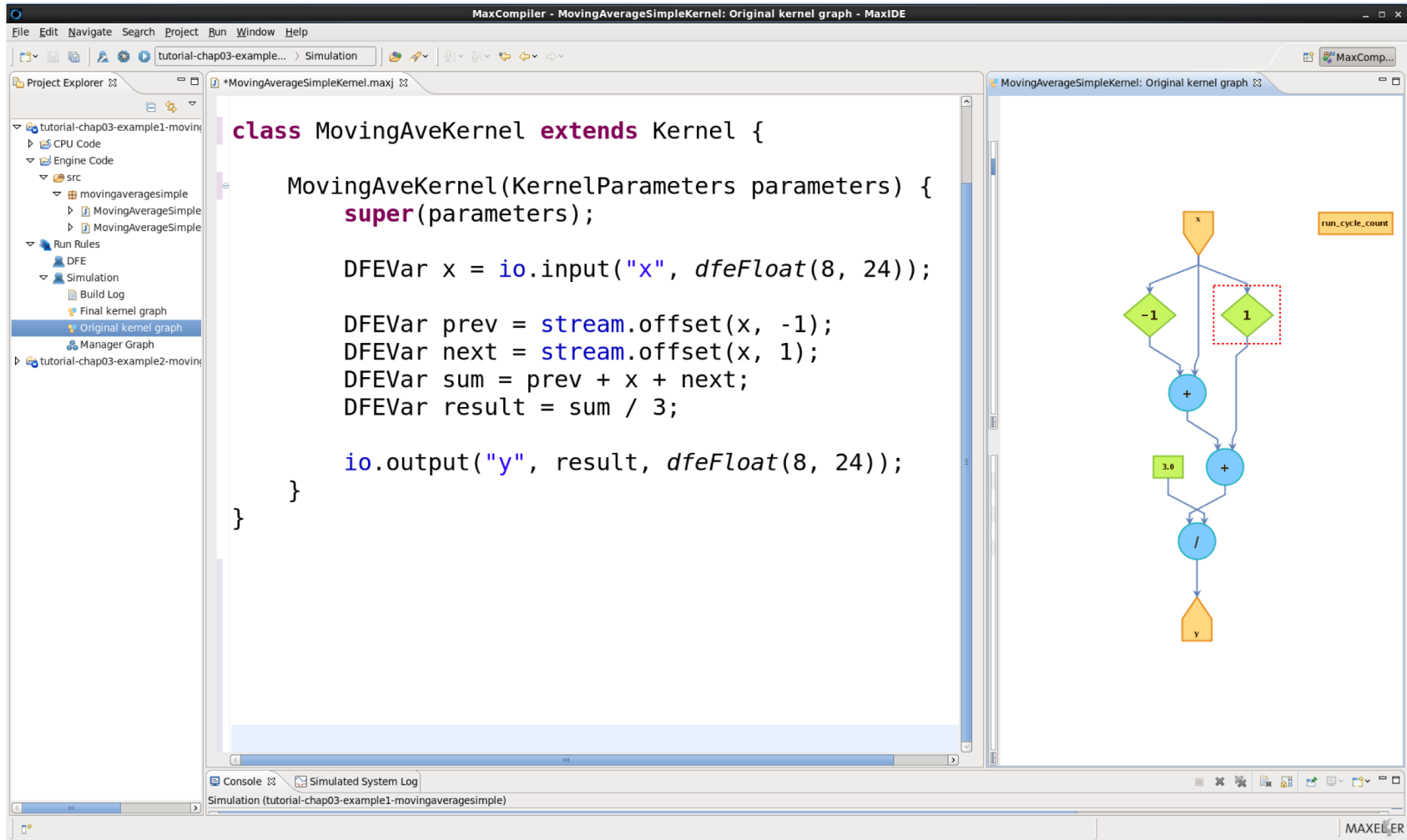
- Connection between operators in the dataflow graph
- An edge of the dataflow graph
- Stream of data elements of a certain type and size
 - (long vector)
- Physically it is a set of wires in the hardware
- It looks like a variable in MaxJ code
- **BUT IS NOT A VARIABLE!** (in traditional CS sense)

MaxJ Basics: Java meta-programming

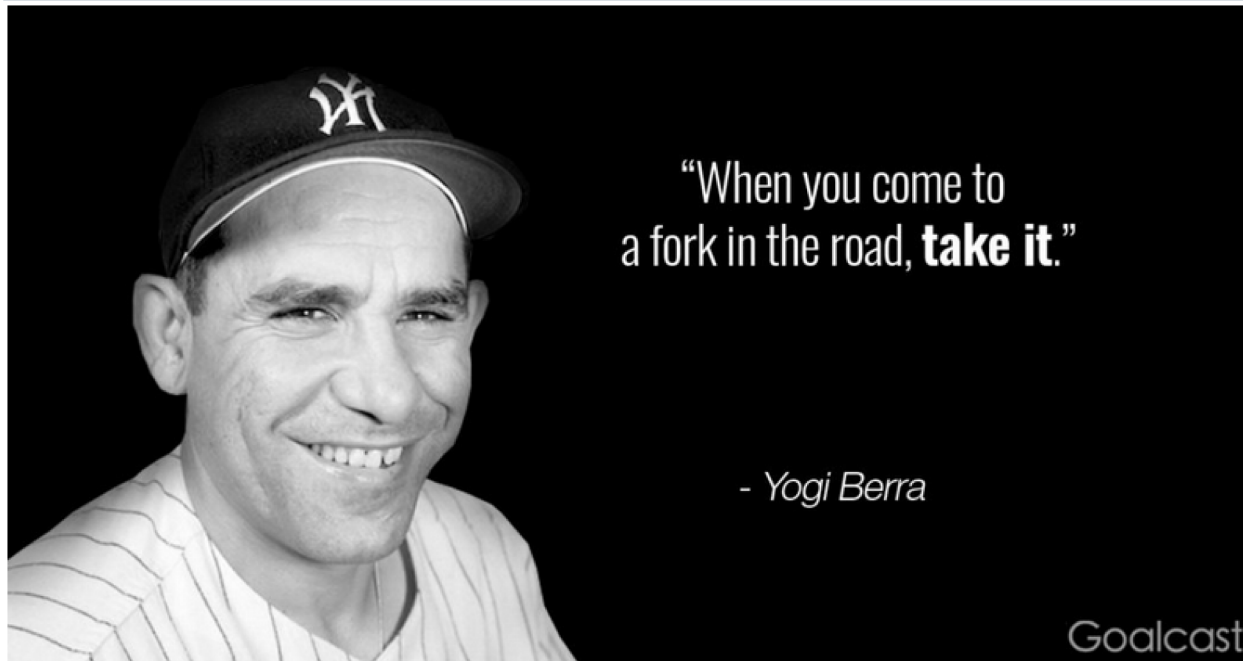
- You can use the full power of Java to write a program that *generates* the dataflow graph
- Java variables can be used as constants in hardware
 - `int y; DFEVar x; x = x + y;`
- Hardware variables can not be read in Java!
 - **Cannot do: `int y; DFEVar x; y = x;`**
- Java conditionals and loops choose 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



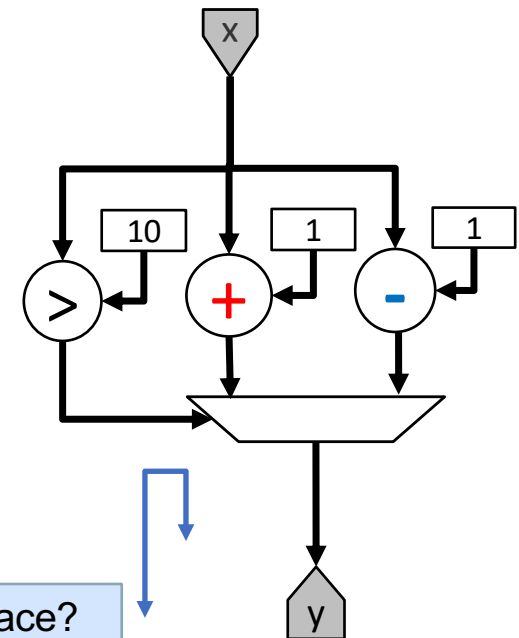
What about branches



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

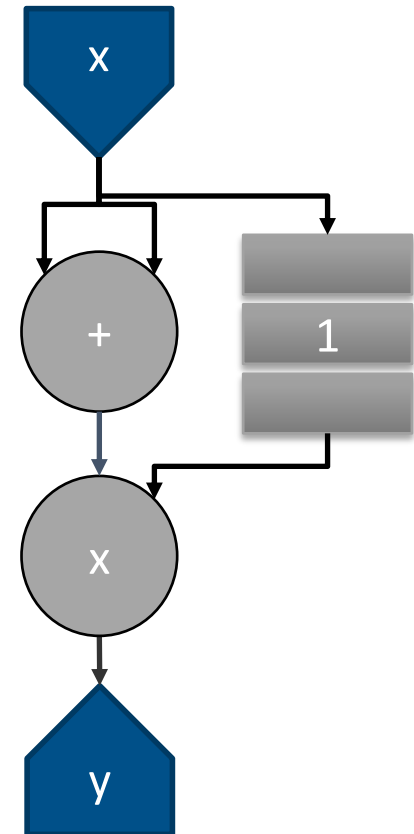
- The dataflow graph in a kernel is statically scheduled and will be executed simultaneously in a parallel fashion
- Operations have inherent latencies
 - If different data paths meet, they need to be balanced and delays (FIFOs) are inserted
 - The scheduler tries to minimise the costs of implementing 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;
```

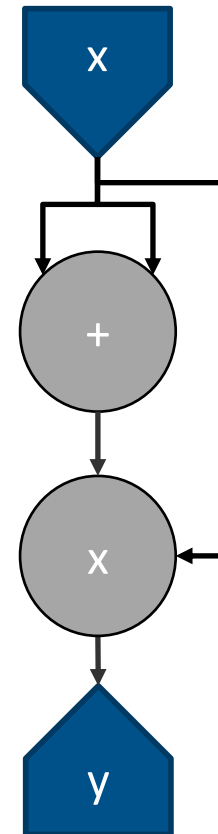
```
y = (x + x) * x;
```

```
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);
```



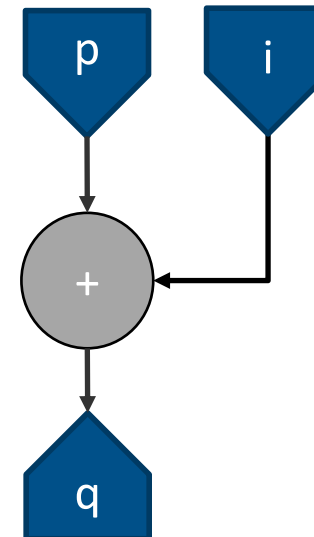
MaxJ Intro: Working with loop counters

- 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

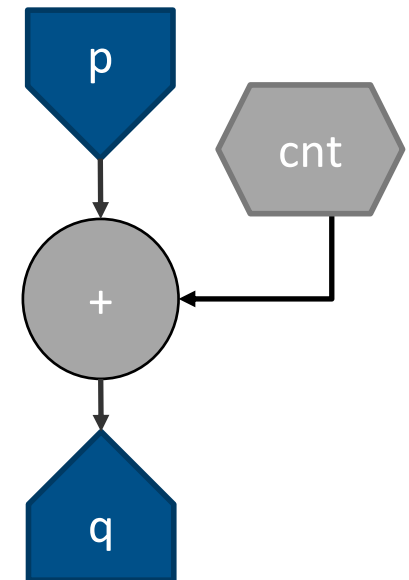
MaxJ Intro: Working with loop counters

- 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));
```

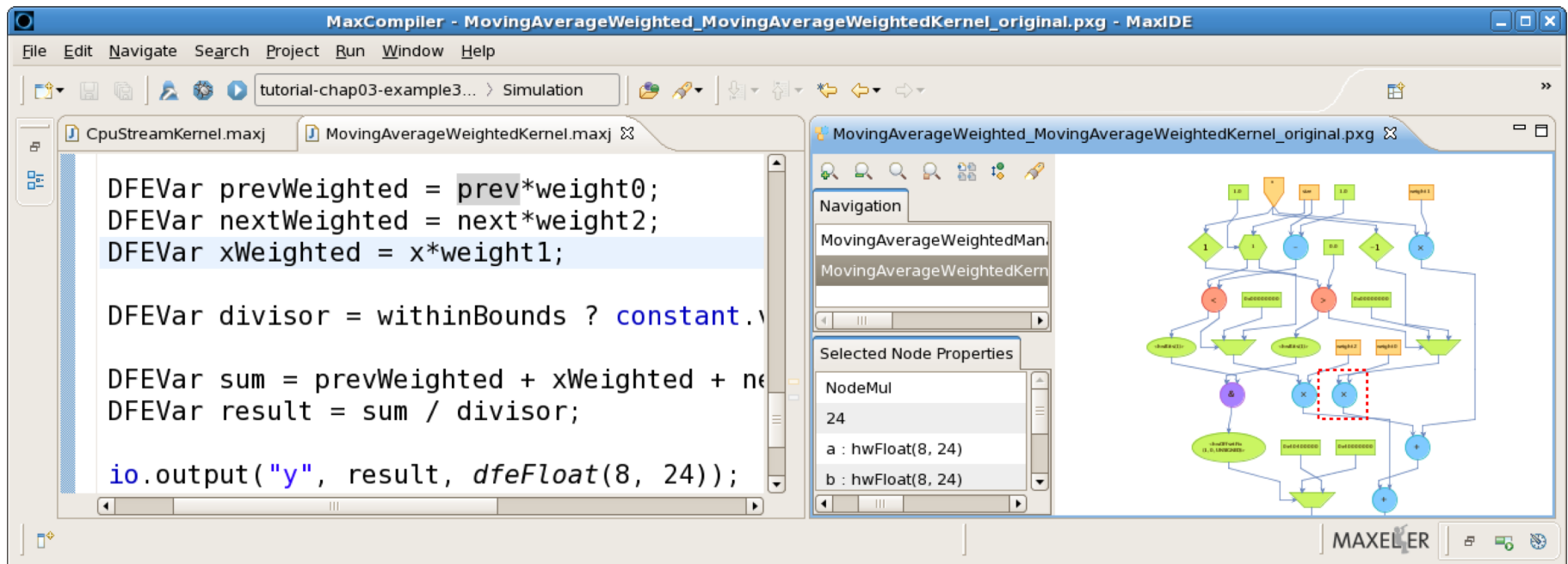
Half as many inputs
Less data transfer

- 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;*



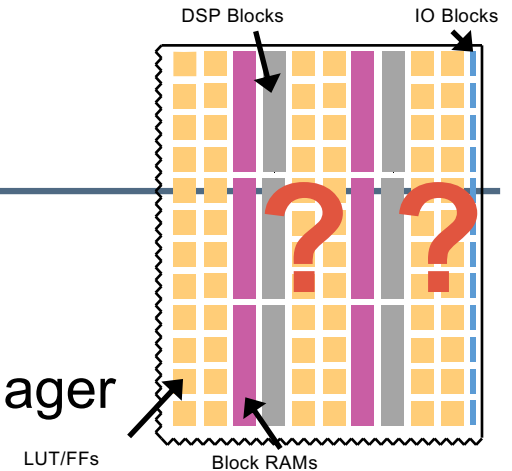
MaxJ Intro: Programming Components

- **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      :  
30      6.4:    d.Sell = bid.Price >= highPrice & order_book.securityId === secId;  
31      :  
32      :  
33      :  
34      :  
35      :  
36      :  
37      :  
38      :  
39      :  
40      :  
41      :  
42      :  
43      :  
44      :  
45      :  
46      :  
47      :  
48      :  
49      :  
50      :  
51      :  
52      :  
53      :  
54      :  
55      :  
56      :  
57      :  
58      :  
59      :  
60      :  
61      :  
62      :  
63      :  
64      :  
65      :  
66      :  
67      :  
68      :  
69      :  
70      :  
71      :  
72      :  
73      :  
74      :  
75      :  
76      :  
77      :  
78      :  
79      :  
80      :  
81      :  
82      :  
83      :  
84      :  
85      :  
86      :  
87      :  
88      :  
89      :  
90      :  
91      :  
92      :  
93      :  
94      :  
95      :  
96      :  
97      :  
98      :  
99      :  
100     :  
101     :  
102     :  
103     :  
104     :  
105     :  
106     :  
107     :  
108     :  
109     :  
110     :  
111     :  
112     :  
113     :  
114     :  
115     :  
116     :  
117     :  
118     :  
119     :  
120     :  
121     :  
122     :  
123     :  
124     :  
125     :  
126     :  
127     :  
128     :  
129     :  
130     :  
131     :  
132     :  
133     :  
134     :  
135     :  
136     :  
137     :  
138     :  
139     :  
140     :  
141     :  
142     :  
143     :  
144     :  
145     :  
146     :  
147     :  
148     :  
149     :  
150     :  
151     :  
152     :  
153     :  
154     :  
155     :  
156     :  
157     :  
158     :  
159     :  
160     :  
161     :  
162     :  
163     :  
164     :  
165     :  
166     :  
167     :  
168     :  
169     :  
170     :  
171     :  
172     :  
173     :  
174     :  
175     :  
176     :  
177     :  
178     :  
179     :  
180     :  
181     :  
182     :  
183     :  
184     :  
185     :  
186     :  
187     :  
188     :  
189     :  
190     :  
191     :  
192     :  
193     :  
194     :  
195     :  
196     :  
197     :  
198     :  
199     :  
200     :  
201     :  
202     :  
203     :  
204     :  
205     :  
206     :  
207     :  
208     :  
209     :  
210     :  
211     :  
212     :  
213     :  
214     :  
215     :  
216     :  
217     :  
218     :  
219     :  
220     :  
221     :  
222     :  
223     :  
224     :  
225     :  
226     :  
227     :  
228     :  
229     :  
230     :  
231     :  
232     :  
233     :  
234     :  
235     :  
236     :  
237     :  
238     :  
239     :  
240     :  
241     :  
242     :  
243     :  
244     :  
245     :  
246     :  
247     :  
248     :  
249     :  
250     :  
251     :  
252     :  
253     :  
254     :  
255     :  
256     :  
257     :  
258     :  
259     :  
260     :  
261     :  
262     :  
263     :  
264     :  
265     :  
266     :  
267     :  
268     :  
269     :  
270     :  
271     :  
272     :  
273     :  
274     :  
275     :  
276     :  
277     :  
278     :  
279     :  
280     :  
281     :  
282     :  
283     :  
284     :  
285     :  
286     :  
287     :  
288     :  
289     :  
290     :  
291     :  
292     :  
293     :  
294     :  
295     :  
296     :  
297     :  
298     :  
299     :  
300     :  
301     :  
302     :  
303     :  
304     :  
305     :  
306     :  
307     :  
308     :  
309     :  
310     :  
311     :  
312     :  
313     :  
314     :  
315     :  
316     :  
317     :  
318     :  
319     :  
320     :  
321     :  
322     :  
323     :  
324     :  
325     :  
326     :  
327     :  
328     :  
329     :  
330     :  
331     :  
332     :  
333     :  
334     :  
335     :  
336     :  
337     :  
338     :  
339     :  
340     :  
341     :  
342     :  
343     :  
344     :  
345     :  
346     :  
347     :  
348     :  
349     :  
350     :  
351     :  
352     :  
353     :  
354     :  
355     :  
356     :  
357     :  
358     :  
359     :  
360     :  
361     :  
362     :  
363     :  
364     :  
365     :  
366     :  
367     :  
368     :  
369     :  
370     :  
371     :  
372     :  
373     :  
374     :  
375     :  
376     :  
377     :  
378     :  
379     :  
380     :  
381     :  
382     :  
383     :  
384     :  
385     :  
386     :  
387     :  
388     :  
389     :  
390     :  
391     :  
392     :  
393     :  
394     :  
395     :  
396     :  
397     :  
398     :  
399     :  
400     :  
401     :  
402     :  
403     :  
404     :  
405     :  
406     :  
407     :  
408     :  
409     :  
410     :  
411     :  
412     :  
413     :  
414     :  
415     :  
416     :  
417     :  
418     :  
419     :  
420     :  
421     :  
422     :  
423     :  
424     :  
425     :  
426     :  
427     :  
428     :  
429     :  
430     :  
431     :  
432     :  
433     :  
434     :  
435     :  
436     :  
437     :  
438     :  
439     :  
440     :  
441     :  
442     :  
443     :  
444     :  
445     :  
446     :  
447     :  
448     :  
449     :  
450     :  
451     :  
452     :  
453     :  
454     :  
455     :  
456     :  
457     :  
458     :  
459     :  
460     :  
461     :  
462     :  
463     :  
464     :  
465     :  
466     :  
467     :  
468     :  
469     :  
470     :  
471     :  
472     :  
473     :  
474     :  
475     :  
476     :  
477     :  
478     :  
479     :  
480     :  
481     :  
482     :  
483     :  
484     :  
485     :  
486     :  
487     :  
488     :  
489     :  
490     :  
491     :  
492     :  
493     :  
494     :  
495     :  
496     :  
497     :  
498     :  
499     :  
500     :  
501     :  
502     :  
503     :  
504     :  
505     :  
506     :  
507     :  
508     :  
509     :  
510     :  
511     :  
512     :  
513     :  
514     :  
515     :  
516     :  
517     :  
518     :  
519     :  
520     :  
521     :  
522     :  
523     :  
524     :  
525     :  
526     :  
527     :  
528     :  
529     :  
530     :  
531     :  
532     :  
533     :  
534     :  
535     :  
536     :  
537     :  
538     :  
539     :  
540     :  
541     :  
542     :  
543     :  
544     :  
545     :  
546     :  
547     :  
548     :  
549     :  
550     :  
551     :  
552     :  
553     :  
554     :  
555     :  
556     :  
557     :  
558     :  
559     :  
560     :  
561     :  
562     :  
563     :  
564     :  
565     :  
566     :  
567     :  
568     :  
569     :  
570     :  
571     :  
572     :  
573     :  
574     :  
575     :  
576     :  
577     :  
578     :  
579     :  
580     :  
581     :  
582     :  
583     :  
584     :  
585     :  
586     :  
587     :  
588     :  
589     :  
590     :  
591     :  
592     :  
593     :  
594     :  
595     :  
596     :  
597     :  
598     :  
599     :  
600     :  
601     :  
602     :  
603     :  
604     :  
605     :  
606     :  
607     :  
608     :  
609     :  
610     :  
611     :  
612     :  
613     :  
614     :  
615     :  
616     :  
617     :  
618     :  
619     :  
620     :  
621     :  
622     :  
623     :  
624     :  
625     :  
626     :  
627     :  
628     :  
629     :  
630     :  
631     :  
632     :  
633     :  
634     :  
635     :  
636     :  
637     :  
638     :  
639     :  
640     :  
641     :  
642     :  
643     :  
644     :  
645     :  
646     :  
647     :  
648     :  
649     :  
650     :  
651     :  
652     :  
653     :  
654     :  
655     :  
656     :  
657     :  
658     :  
659     :  
660     :  
661     :  
662     :  
663     :  
664     :  
665     :  
666     :  
667     :  
668     :  
669     :  
670     :  
671     :  
672     :  
673     :  
674     :  
675     :  
676     :  
677     :  
678     :  
679     :  
680     :  
681     :  
682     :  
683     :  
684     :  
685     :  
686     :  
687     :  
688     :  
689     :  
690     :  
691     :  
692     :  
693     :  
694     :  
695     :  
696     :  
697     :  
698     :  
699     :  
700     :  
701     :  
702     :  
703     :  
704     :  
705     :  
706     :  
707     :  
708     :  
709     :  
710     :  
711     :  
712     :  
713     :  
714     :  
715     :  
716     :  
717     :  
718     :  
719     :  
720     :  
721     :  
722     :  
723     :  
724     :  
725     :  
726     :  
727     :  
728     :  
729     :  
730     :  
731     :  
732     :  
733     :  
734     :  
735     :  
736     :  
737     :  
738     :  
739     :  
740     :  
741     :  
742     :  
743     :  
744     :  
745     :  
746     :  
747     :  
748     :  
749     :  
750     :  
751     :  
752     :  
753     :  
754     :  
755     :  
756     :  
757     :  
758     :  
759     :  
760     :  
761     :  
762     :  
763     :  
764     :  
765     :  
766     :  
767     :  
768     :  
769     :  
770     :  
771     :  
772     :  
773     :  
774     :  
775     :  
776     :  
777     :  
778     :  
779     :  
780     :  
781     :  
782     :  
783     :  
784     :  
785     :  
786     :  
787     :  
788     :  
789     :  
790     :  
791     :  
792     :  
793     :  
794     :  
795     :  
796     :  
797     :  
798     :  
799     :  
800     :  
801     :  
802     :  
803     :  
804     :  
805     :  
806     :  
807     :  
808     :  
809     :  
810     :  
811     :  
812     :  
813     :  
814     :  
815     :  
816     :  
817     :  
818     :  
819     :  
820     :  
821     :  
822     :  
823     :  
824     :  
825     :  
826     :  
827     :  
828     :  
829     :  
830     :  
831     :  
832     :  
833     :  
834     :  
835     :  
836     :  
837     :  
838     :  
839     :  
840     :  
841     :  
842     :  
843     :  
844     :  
845     :  
846     :  
847     :  
848     :  
849     :  
850     :  
851     :  
852     :  
853     :  
854     :  
855     :  
856     :  
857     :  
858     :  
859     :  
860     :  
861     :  
862     :  
863     :  
864     :  
865     :  
866     :  
867     :  
868     :  
869     :  
870     :  
871     :  
872     :  
873     :  
874     :  
875     :  
876     :  
877     :  
878     :  
879     :  
880     :  
881     :  
882     :  
883     :  
884     :  
885     :  
886     :  
887     :  
888     :  
889     :  
890     :  
891     :  
892     :  
893     :  
894     :  
895     :  
896     :  
897     :  
898     :  
899     :  
900     :  
901     :  
902     :  
903     :  
904     :  
905     :  
906     :  
907     :  
908     :  
909     :  
910     :  
911     :  
912     :  
913     :  
914     :  
915     :  
916     :  
917     :  
918     :  
919     :  
920     :  
921     :  
922     :  
923     :  
924     :  
925     :  
926     :  
927     :  
928     :  
929     :  
930     :  
931     :  
932     :  
933     :  
934     :  
935     :  
936     :  
937     :  
938     :  
939     :  
940     :  
941     :  
942     :  
943     :  
944     :  
945     :  
946     :  
947     :  
948     :  
949     :  
950     :  
951     :  
952     :  
953     :  
954     :  
955     :  
956     :  
957     :  
958     :  
959     :  
960     :  
961     :  
962     :  
963     :  
964     :  
965     :  
966     :  
967     :  
968     :  
969     :  
970     :  
971     :  
972     :  
973     :  
974     :  
975     :  
976     :  
977     :  
978     :  
979     :  
980     :  
981     :  
982     :  
983     :  
984     :  
985     :  
986     :  
987     :  
988     :  
989     :  
990     :  
991     :  
992     :  
993     :  
994     :  
995     :  
996     :  
997     :  
998     :  
999     :  
1000    :  
1001    :  
1002    :  
1003    :  
1004    :  
1005    :  
1006    :  
1007    :  
1008    :  
1009    :  
1010    :  
1011    :  
1012    :  
1013    :  
1014    :  
1015    :  
1016    :  
1017    :  
1018    :  
1019    :  
1020    :  
1021    :  
1022    :  
1023    :  
1024    :  
1025    :  
1026    :  
1027    :  
1028    :  
1029    :  
1030    :  
1031    :  
1032    :  
1033    :  
1034    :  
1035    :  
1036    :  
1037    :  
1038    :  
1039    :  
1040    :  
1041    :  
1042    :  
1043    :  
1044    :  
1045    :  
1046    :  
1047    :  
1048    :  
1049    :  
1050    :  
1051    :  
1052    :  
1053    :  
1054    :  
1055    :  
1056    :  
1057    :  
1058    :  
1059    :  
1060    :  
1061    :  
1062    :  
1063    :  
1064    :  
1065    :  
1066    :  
1067    :  
1068    :  
1069    :  
1070    :  
1071    :  
1072    :  
1073    :  
1074    :  
1075    :  
1076    :  
1077    :  
1078    :  
1079    :  
1080    :  
1081    :  
1082    :  
1083    :  
1084    :  
1085    :  
1086    :  
1087    :  
1088    :  
1089    :  
1090    :  
1091    :  
1092    :  
1093    :  
1094    :  
1095    :  
1096    :  
1097    :  
1098    :  
1099    :  
1100    :  
1101    :  
1102    :  
1103    :  
1104    :  
1105    :  
1106    :  
1107    :  
1108    :  
1109    :  
1110    :  
1111    :  
1112    :  
1113    :  
1114    :  
1115    :  
1116    :  
1117    :  
1118    :  
1119    :  
1120    :  
1121    :  
1122    :  
1123    :  
1124    :  
1125    :  
1126    :  
1127    :  
1128    :  
1129    :  
1130    :  
1131    :  
1132    :  
1133    :  
1134    :  
1135    :  
1136    :  
1137    :  
1138    :  
1139    :  
1140    :  
1141    :  
1142    :  
1143    :  
1144    :  
1145    :  
1146    :  
1147    :  
1148    :  
1149    :  
1150    :  
1151    :  
1152    :  
1153    :  
1154    :  
1155    :  
1156    :  
1157    :  
1158    :  
1159    :  
1160    :  
1161    :  
1162    :  
1163    :  
1164    :  
1165    :  
1166    :  
1167    :  
1168    :  
1169    :  
1170    :  
1171    :  
1172    :  
1173    :  
1174    :  
1175    :  
1176    :  
1177    :  
1178    :  
1179    :  
1180    :  
1181    :  
1182    :  
1183    :  
1184    :  
1185    :  
1186    :  
1187    :  
1188    :  
1189    :  
1190    :  
1191    :  
1192    :  
1193    :  
1194    :  
1195    :  
1196    :  
1197    :  
1198    :  
1199    :  
1200    :  
1201    :  
1202    :  
1203    :  
1204    :  
1205    :  
1206    :  
1207    :  
1208    :  
1209    :  
1210    :  
1211    :  
1212    :  
1213    :  
1214    :  
1215    :  
1216    :  
1217    :  
1218    :  
1219    :  
1220    :  
1221    :  
1222    :  
1223    :  
1224    :  
1225    :  
1226    :  
1227    :  
1228    :  
1229    :  
1230    :  
1231    :  
1232    :  
1233    :  
1234    :  
1235    :  
1236    :  
1237    :  
1238    :  
1239    :  
1240    :  
1241    :  
1242    :  
1243    :  
1244    :  
1245    :  
1246    :  
1247    :  
1248    :  
1249    :  
1250    :  
1251    :  
1252    :  
1253    :  
1254    :  
1255    :  
1256    :  
1257    :  
1258    :  
1259    :  
1260    :  
1261    :  
1262    :  
1263    :  
1264    :  
1265    :  
1266    :  
1267    :  
1268    :  
1269    :  
1270    :  
1271    :  
1272    :  
1273    :  
1274    :  
1275    :  
1276    :  
1277    :  
1278    :  
1279    :  
1280    :  
1281    :  
1282    :  
1283    :  
1284    :  
1285    :  
1286    :  
1287    :  
1288    :  
1289    :  
1290    :  
1291    :  
1292    :  
1293    :  
1294    :  
1295    :  
1296    :  
1297    :  
1298    :  
1299    :  
1300    :  
1301    :  
1302    :  
1303    :  
1304    :  
1305    :  
1306    :  
1307    :  
1308    :  
1309    :  
1310    :  
1311    :  
1312    :  
1313    :  
1314    :  
1315    :  
1316    :  
1317    :  
1318    :  
1319    :  
1320    :  
1321    :  
1322    :  
1323    :  
1324    :  
1325    :  
1326    :  
1327    :  
1328    :  
1329    :  
1330    :  
1331    :  
1332    :  
1333    :  
1334    :  
1335    :  
1336    :  
1337    :  
1338    :  
1339    :  
1340    :  
1341    :  
1342    :  
1343    :  
1344    :  
1345    :  
1346    :  
1347    :  
1348    :  
1349    :  
1350    :  
1351    :  
1352    :  
1353    :  
1354    :  
1355    :  
1356    :  
1357    :  
1358    :  
1359    :  
1360    :  
1361    :  
1362    :  
1363    :  
1364    :  
1365    :  
1366    :  
1367    :  
1368    :  
1369    :  
1370    :  
1371    :  
1372    :  
1373    :  
1374    :  
1375    :  
1376    :  
1377    :  
1378    :  
1379    :  
1380    :  
1381    :  
1382    :  
1383    :  
1384    :  
1385    :  
1386    :  
1387    :  
1388    :  
1389    :  
1390    :  
1391    :  
1392    :  
1393    :  
1394    :  
1395    :  
1396    :  
1397    :  
1398    :  
1399    :  
1400    :  
1401    :  
1402    :  
1403    :  
1404    :  
1405    :  
1406    :  
1407    :  
1408    :  
1409    :  
1410    :  
1411    :  
1412    :  
1413    :  
1414    :  
1415    :  
1416    :  
1417    :  
1418    :  
1419    :  
1420    :  
1421    :  
1422    :  
1423    :  
1424    :  
1425    :  
1426    :  
1427    :  
1428    :  
1429    :  
1430    :  
1431    :  
1432    :  
1433    :  
1434    :  
1435    :  
1436    :  
1437    :  
1438    :  
1439    :  
1440    :  
1441    :  
1442    :  
1443    :  
1444    :  
1445    :  
1446    :  
1447    :  
1448    :  
1449    :  
1450    :  
1451    :  
1452    :  
1453    :  
1454    :  
1455    :  
1456    :  
1457    :  
1458    :  
1459    :  
1460    :  
1461    :  
1462    :  
1463    :  
1464    :  
1465    :  
1466    :  
1467    :  
1468    :  
1469    :  
1470    :  
1471    :  
1472    :  
1473    :  
1474    :  
1475    :  
1476    :  
1477    :  
1478    :  
1479    :  
1480    :  
1481    :  
1482    :  
1483    :  
1484    :  
1485    :  
1486    :  
1487    :  
1488    :  
1489    :  
1490    :  
1491    :  
1492    :  
1493    :  
1494    :  
1495    :  
1496    :  
1497    :  
1498    :  
1499    :  

```

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

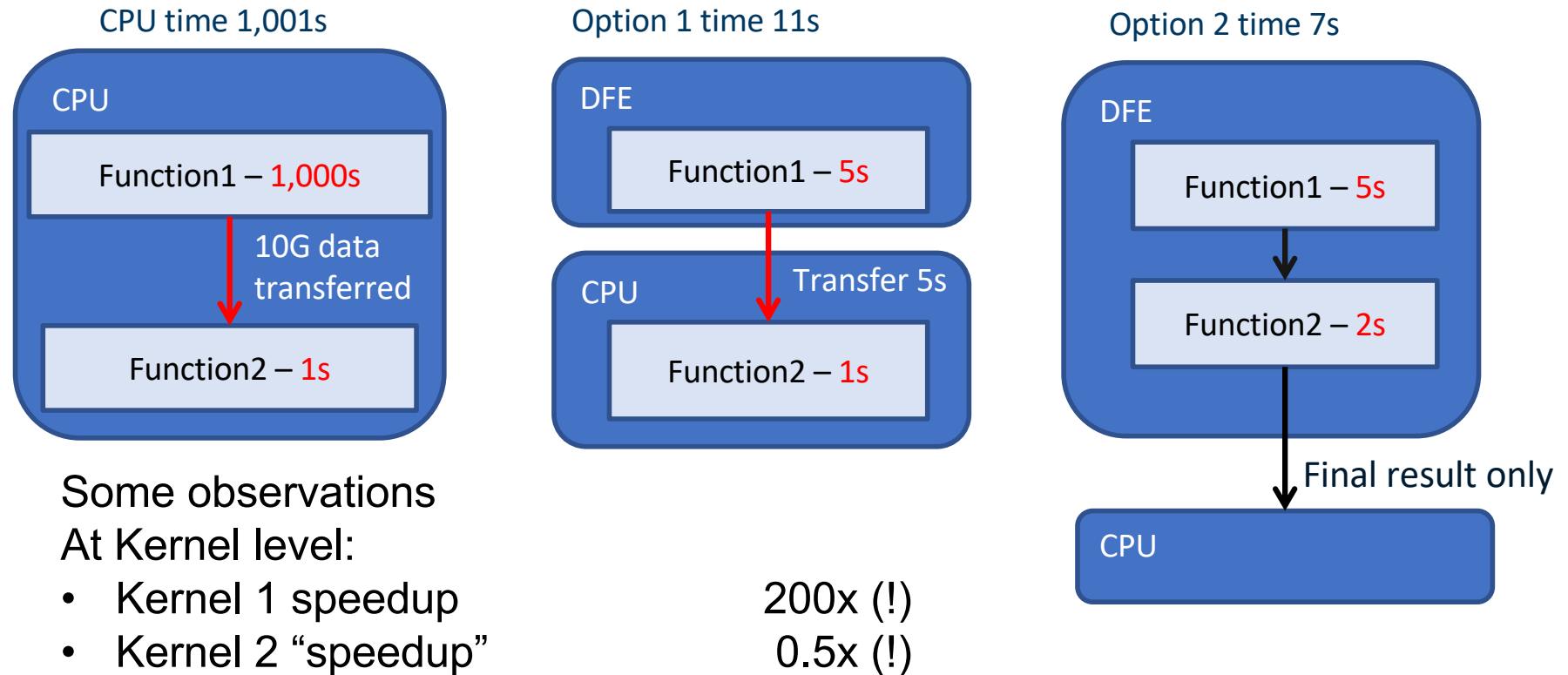
VHDL vs MaxJ example (HMS CERN)

```
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 DOWNT0 0 );
BEGIN
  PROCESS( clk )
  BEGIN
    IF( RISING_EDGE( clk ) ) THEN
      IF( NOT jetIn1.DataValid ) THEN
        jetOut <= cEmptyJet;
      ELSE
        jetOut.Energy( 15 DOWNT0 0 ) <=
          jetIn1.Energy + jetIn2.Energy;
        jetOut.Ecal( 15 DOWNT0 0 ) <=
          jetIn1.Ecal + jetIn2.Ecal;
        jetOut.DataValid <= TRUE;
      END IF;
    END IF;
  END PROCESS;
END ARCHITECTURE behavioral;
```

System Example: Decelerate to Accelerate



At System level:

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

But what about the required effort?

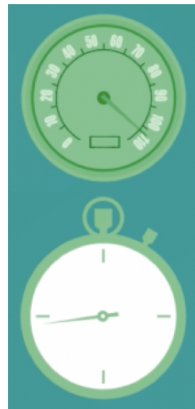
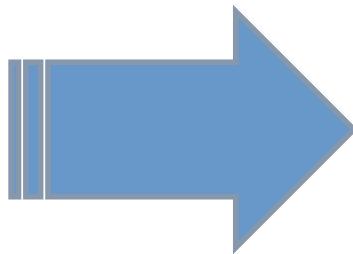
Easy it is not (and not really new)

Slotnick's law (of effort):

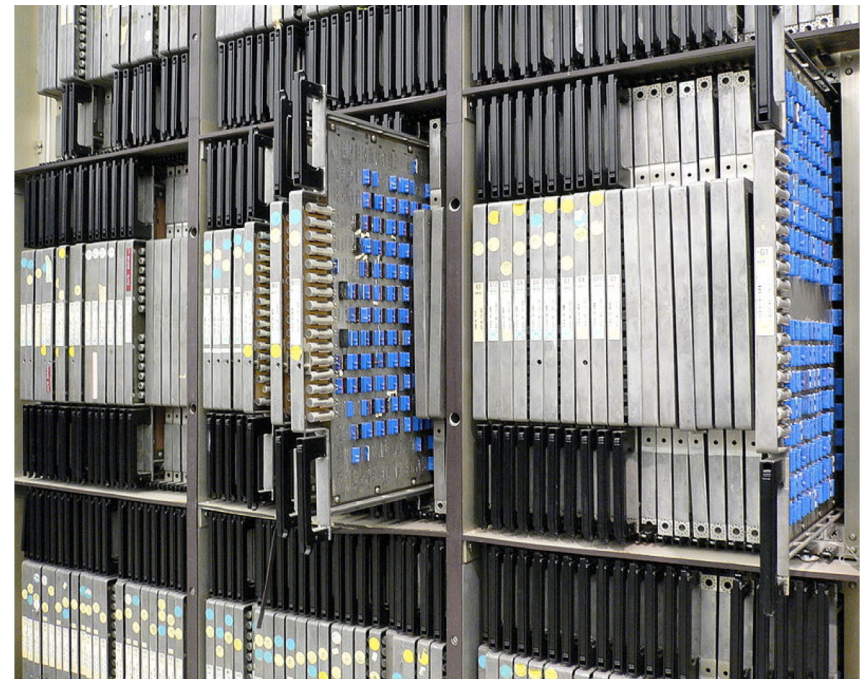
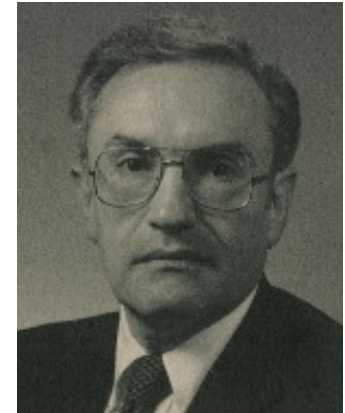
“The parallel approach to computing does require that some **original thinking** be done **about numerical analysis and data management** in order to secure efficient use.

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

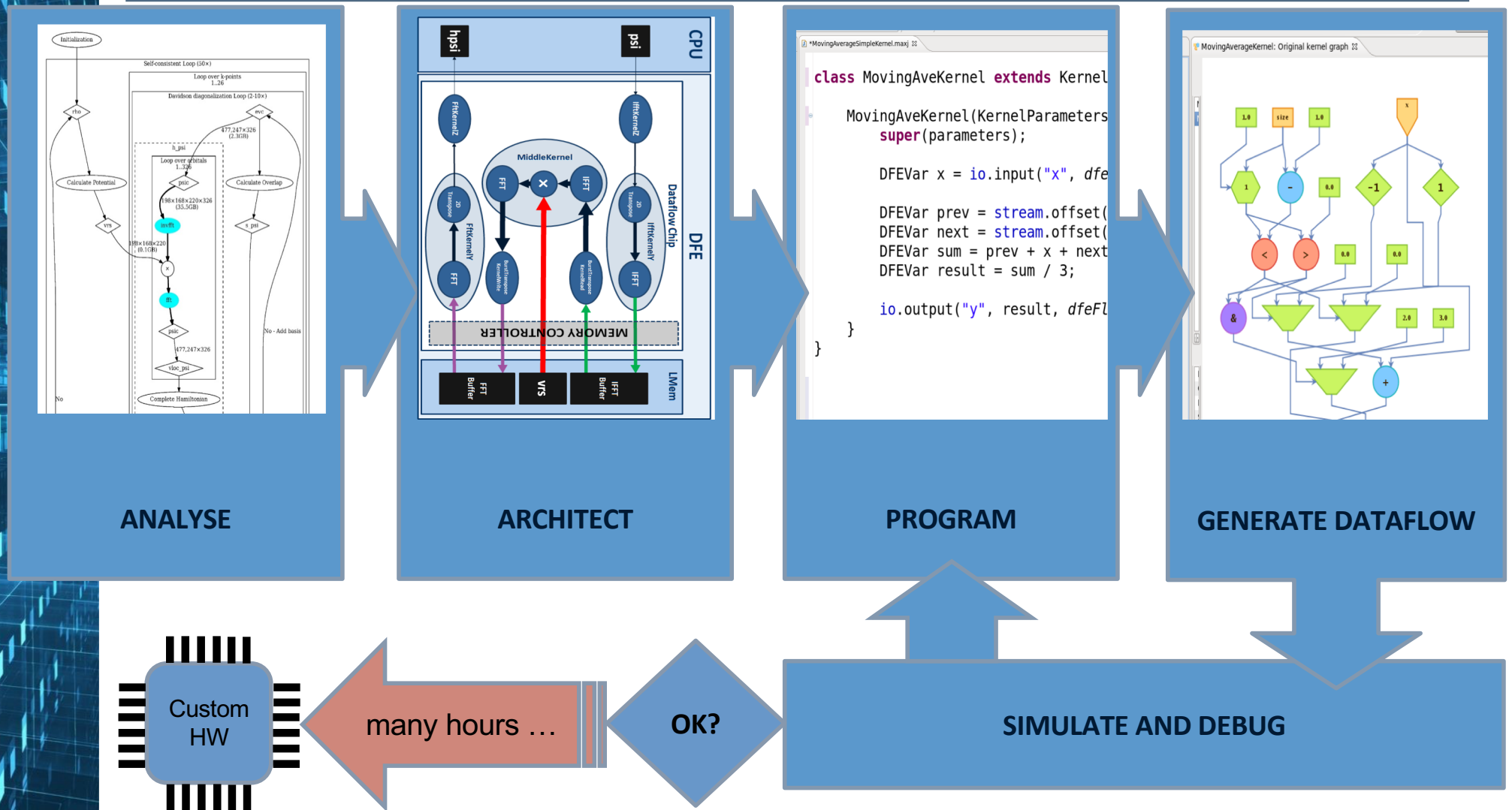
Daniel Slotnick (1931-1985)
Chief Architect of Illiac IV



Turn brain power into performance



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

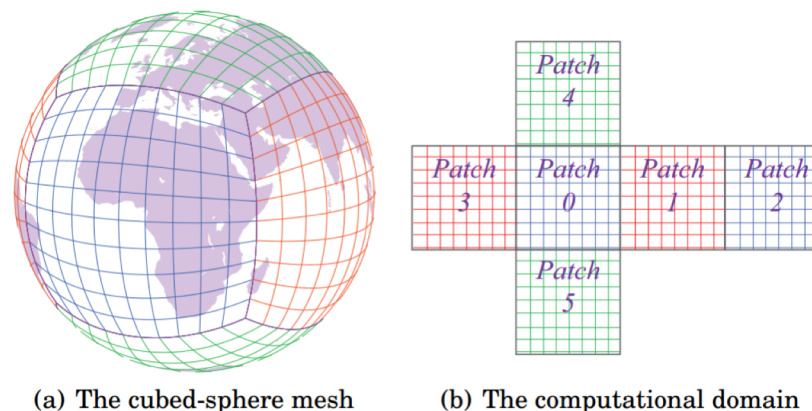


Fig. 1. Mesh and computational domain.

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



Imperial College
London



清华大学
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



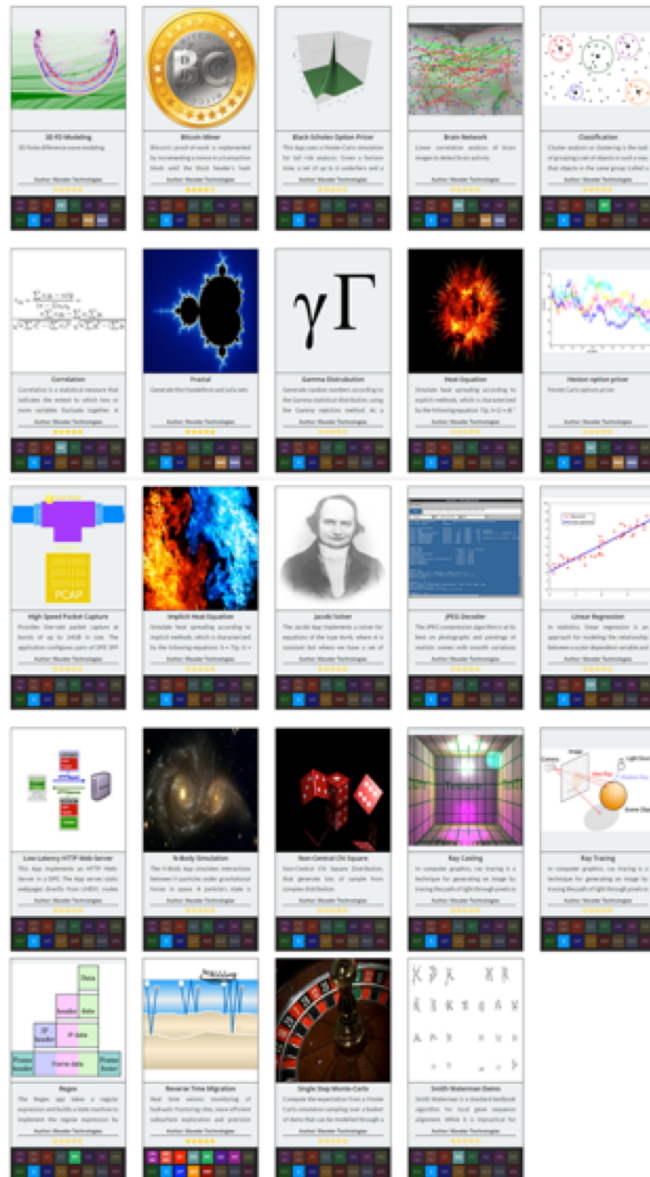
Over 150 Maxeler University Program Members



Maxeler 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):

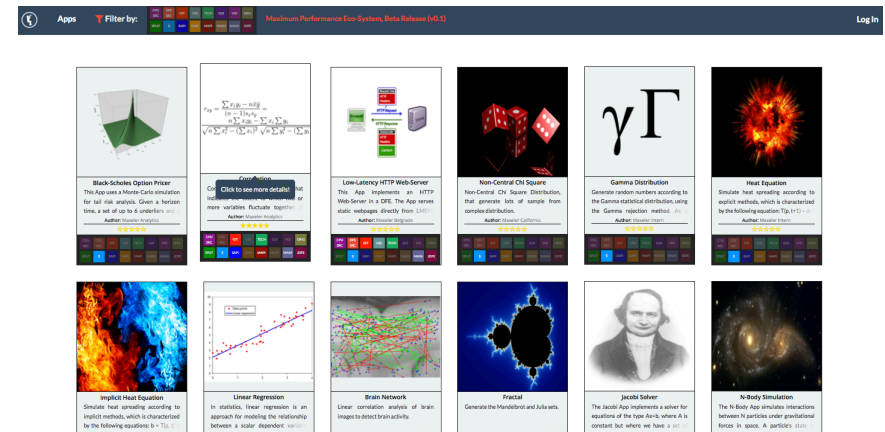
<http://appgallery.maxeler.com>

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

